





λ -calcul pour l'agrégation

Jean Goubault-Larrecq
1. définitions,
confluence,
terminaison

Références bibliographiques

- ❖  Henk Barendregt
The Lambda Calculus — Its Syntax and Semantics
North-Holland (revised ed., 1984)
l'encyclopédie! contient tout
- ❖  Jean-Louis Krivine
Lambda-calcul, types et modèles [seuls les chapitres 1 et 2 sont au programme]
Masson (1991)
version anglaise en ligne: <https://www.irif.fr/~krivine/articles/Lambda.pdf>
- ❖  Richard Lassaigne et Michel de Rougemont
Logique et fondements de l'informatique:
logique du 1er ordre, calculabilité et lambda-calcul
Hermès (1993)
- ❖  Thérèse Hardin-Accart, *cours de lambda-calcul*, DEA SPP, 2003-04
<http://www-spi.lip6.fr/~hardin/DEA/poly.pdf>
- ❖  JGL, *cours de L3 « logique informatique — lambda-calcul »*, ENS Paris-Saclay,
https://projects.lsv.fr/agreg/?page_id=113

Motivation

Alonzo Church

- ❖ Le λ -calcul a été inventé par Alonzo Church en 1932



Alonzo Church

By Princeton University, Fair use, <https://en.wikipedia.org/w/index.php?curid=6082269>

Lisp

History of Lisp

John McCarthy
Artificial Intelligence Laboratory
Stanford University

12 February 1979

This draft gives insufficient mention to many people who helped implement LISP and who contributed ideas. Suggestions for improvements in directions are particularly welcome. Facts about the history of FUNARCO uplevel addressing generally are especially needed.

❖ Le tout premier langage fonctionnel:

📖 John McCarthy *et al.*

LISP 1.5 Programmer's Manual

MIT Press (1962)

```
(define fact(x)
  (cond (eq x 0)
        1
        (* x (fact (- x 1)))))
```

❖ Lisp est un lambda-calcul **enrichi**

(avec des primitives: eq, cond, *, 0, 1, -, etc.)

Lisp

History of Lisp

John McCarthy
Artificial Intelligence Laboratory
Stanford University

12 February 1979

This draft gives insufficient mention to many people who helped implement LISP and who contributed ideas. Suggestions for improvements in directions are particularly welcome. Facts about the history of FUNARCO uplevel addressing generally are especially needed.

❖ Le tout premier langage fonctionnel:

📖 John McCarthy *et al.*

LISP 1.5 Programmer's Manual

MIT Press (1962)

```
(define fact(x)
  (cond (eq x 0)
        1
        (* x (fact (- x 1)))))
```

Langage fonctionnel:
on calcule en **appliquant** des fonctions
à des arguments (eq, cond, *, fact, -)

❖ Lisp est un lambda-calcul **enrichi**
(avec des primitives: eq, cond, *, 0, 1, -, etc.)

Lisp

History of Lisp

John McCarthy
Artificial Intelligence Laboratory
Stanford University

12 February 1979

This draft gives insufficient mention to many people who helped implement LISP and who contributed ideas. Suggestions for improvements in directions are particularly welcome. Facts about the history of FUNARCO uplevel addressing generally are especially needed.

❖ Le tout premier langage fonctionnel:

📖 John McCarthy *et al.*

LISP 1.5 Programmer's Manual

MIT Press (1962)

❖ `(define fact(x)`

`(cond (eq x 0)`

`1`

On peut aussi
définir des fonctions

`(* x (fact (- x 1))))))`

Langage fonctionnel:
on calcule en **appliquant** des fonctions
à des arguments (eq, cond, *, fact, -)

❖ Lisp est un lambda-calcul **enrichi**

(avec des primitives: eq, cond, *, 0, 1, -, etc.)

Lisp

History of Lisp

John McCarthy
Artificial Intelligence Laboratory
Stanford University

12 February 1979

This draft gives insufficient mention to many people who helped implement LISP and who contributed ideas. Suggestions for improvements in directions are particularly welcome. Facts about the history of FUNARCO uplevel addressing generally are especially needed.

- ❖ Le tout premier langage fonctionnel:

📖 John McCarthy *et al.*

LISP 1.5 Programmer's Manual, MIT Press (1962)

- ❖ `(define fact`

`(lambda (x)`

`(cond (eq x 0)`

`1`

`(* x (fact (-x 1))))))`

On peut même définir des fonctions **anonymes**

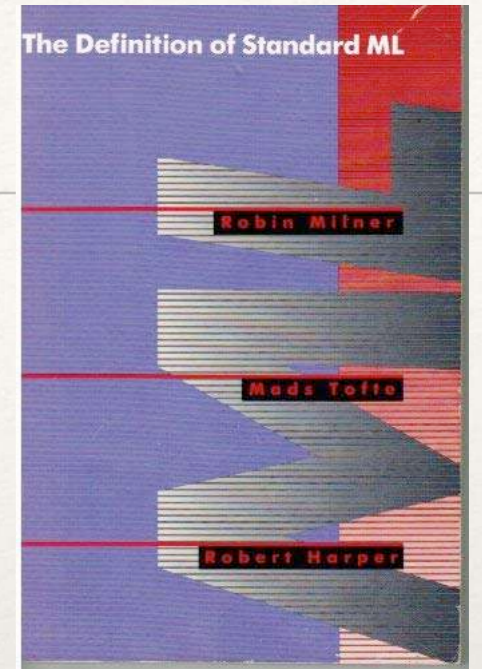
Ceci est directement inspiré de la notation $\lambda x \dots$ du λ -calcul

- ❖ `(mapcar (lambda (x) (+ x 1))`

`(list 1 2 3))`

`; calcule (2 3 4)`

ML



❖ Dû à Robin Milner (1978)

λ -calc. \rightarrow Hope \rightarrow ML \rightarrow CaML \rightarrow CaML light \rightarrow OCaml
... aussi \rightarrow Standard ML (SML/NJ)

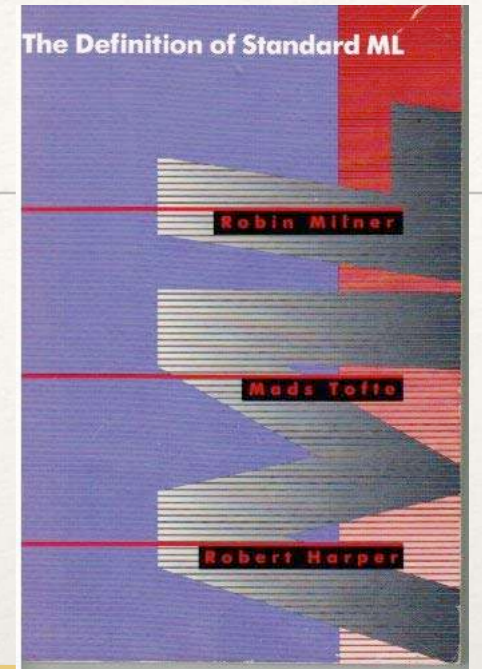
<https://pictures.abebooks.com/isbn/9780262631327-us.jpg>

❖ `fun fact x =
 if x=0
 then 1
 else x*fact(x-1);`

Langage fonctionnel:
on calcule en **appliquant** des fonctions
à des arguments (=, if, *, fact, -);
la syntaxe est simplement plus agréable

On peut aussi **définir** des fonctions — récursives, comme en Lisp

ML



- ❖ Dû à Robin Milner (1978)

λ -calc. \rightarrow Hope \rightarrow ML \rightarrow CaML \rightarrow CaML light \rightarrow OCaml
... aussi \rightarrow Standard ML (SML/NJ)

- ❖ `val rec fact =`

```
fn x => if x=0  
      then 1  
      else x*fact(x-1);
```

On peut même définir des
fonctions **anonymes**

Ceci est directement inspiré de la
notation $\lambda x . \dots$ du λ -calcul

- ❖ `map (fn x => x+1) [1, 2, 3];`
`(* calcule [2, 3, 4] *)`

Haskell



Par Thought up by Darrin Thompson
and produced by Jeff Wheeler –
Thompson-Wheeler logo on the haskell wiki,
Domaine public,
<https://commons.wikimedia.org/w/index.php?curid=8479507>

❖ (1990)

λ -calc. \rightarrow Miranda \rightarrow Haskell

❖ `fact 0 = 1`

`fact x = x*fact(x-1)`

Langage fonctionnel:
on calcule en **appliquant** des fonctions
à des arguments (`*`, `fact`, `-`)

On peut aussi **définir** des fonctions — récursives, comme en Lisp et en ML

Haskell



Par Thought up by Darrin Thompson
...neler -
kell wiki,
hp?curid=8479507

On peut même définir des
fonctions **anonymes**

❖ (1990)

λ -calc. \rightarrow Miranda \rightarrow Haskell

Ceci est directement inspiré de la
notation $\lambda x \dots$ du λ -calcul

```
❖ fact = \x | x=0 -> 1  
        | otherwise -> x*fact(x-1)
```

```
❖ map (\x -> x+1) [1,2,3]  
    -- calcule [2,3,4]
```

```
❖ nat = 0:map (\x -> x+1) nat  
    -- calcule [0, 1, 2, ...]
```

Evaluation paresseuse:
(appel par nom, voir stratégies, plus tard dans le cours)
les arguments de fonction
(ici, :) ne sont évalués que si
on a besoin de les connaître

La syntaxe du λ -calcul

La syntaxe du λ -calcul

- ❖ Très très simple! Les termes sont:

$s, t, u, v, \dots ::=$

x, y, z, \dots variables (en nb. ∞ dénombrable)

| st application (de s à t)

| $\lambda x . s$ λ -abstraction (`fun x -> s`, en Caml)

- ❖ C'est tout! Pas d'entiers, pas de listes, pas de récursion, pas de types, pas de modules, rien d'autre...

La syntaxe du λ -calcul

- ❖ Très très simple! Les termes sont:

$s, t, u, v, \dots ::=$

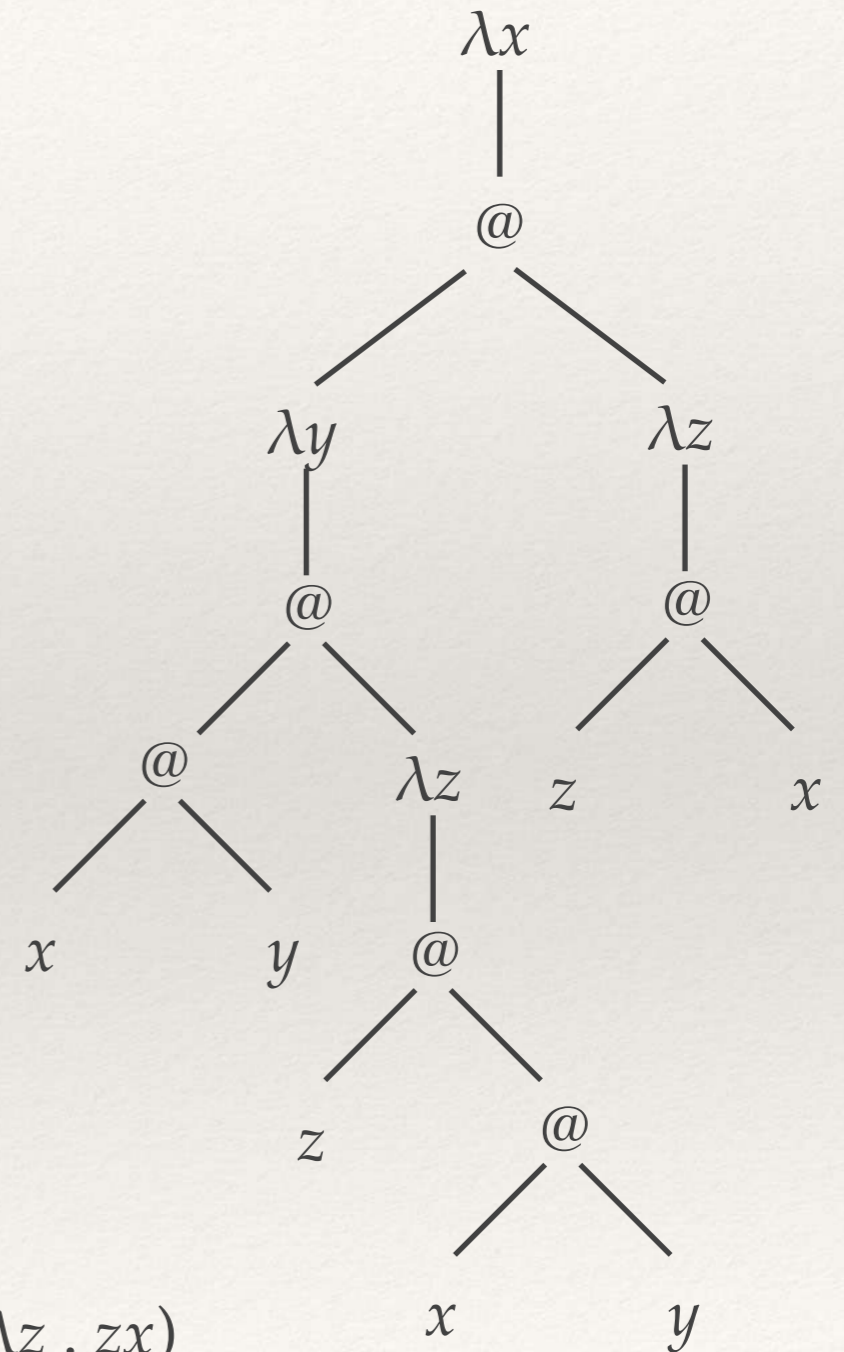
x, y, z, \dots variables (en nb. ∞ dénombrable)

| st application (de s à t)

| $\lambda x . s$ λ -abstraction (`fun x -> s`, en Caml)

- ❖ C'est tout! Pas d'entiers, pas de listes, pas de récursion, pas de types, pas de modules, rien d'autre...
- ❖ Et pourtant, on verra que le langage est Turing-complet

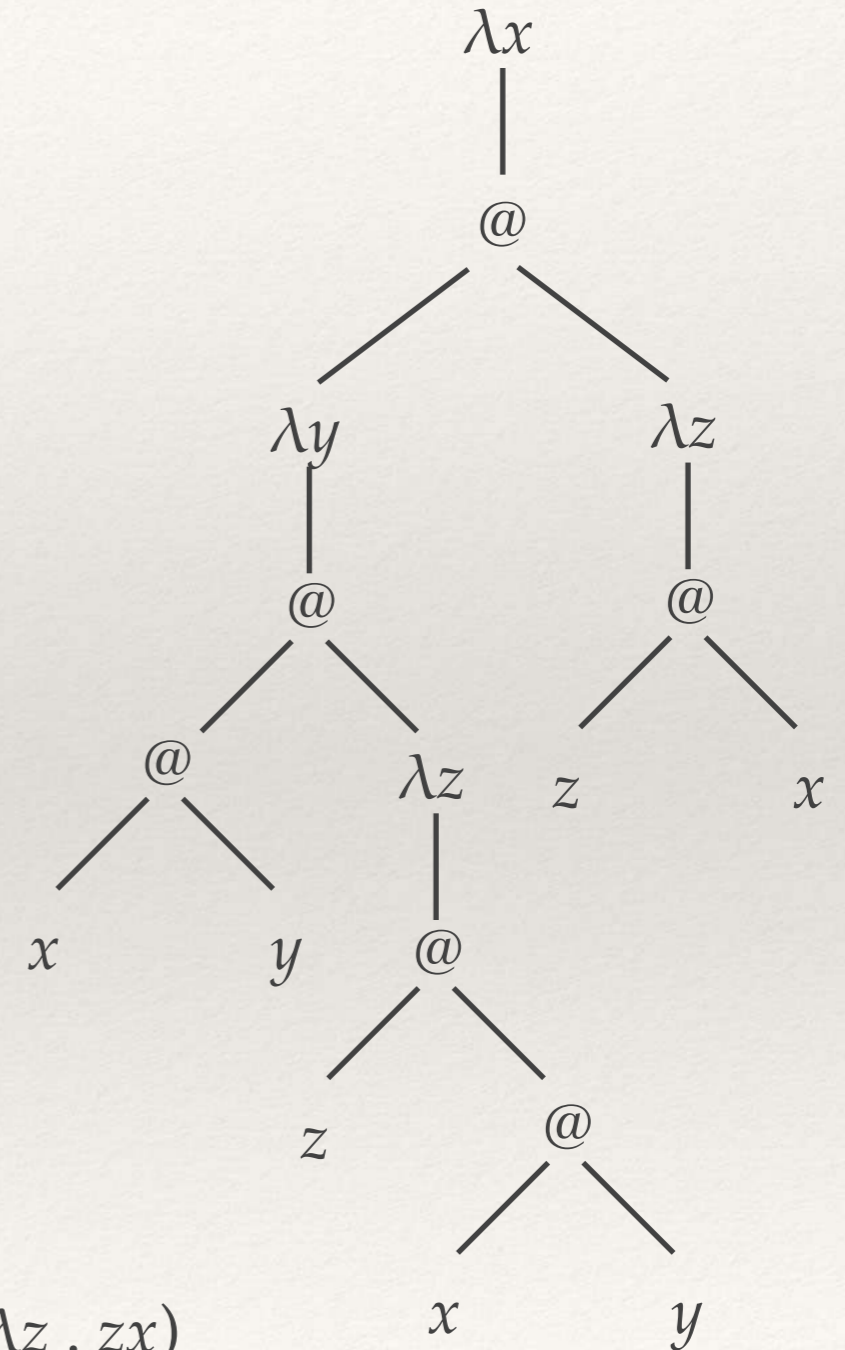
La syntaxe du λ -calcul



$\lambda x . (\lambda y . xy(\lambda z . z(xy))) (\lambda z . zx)$

La syntaxe du λ -calcul

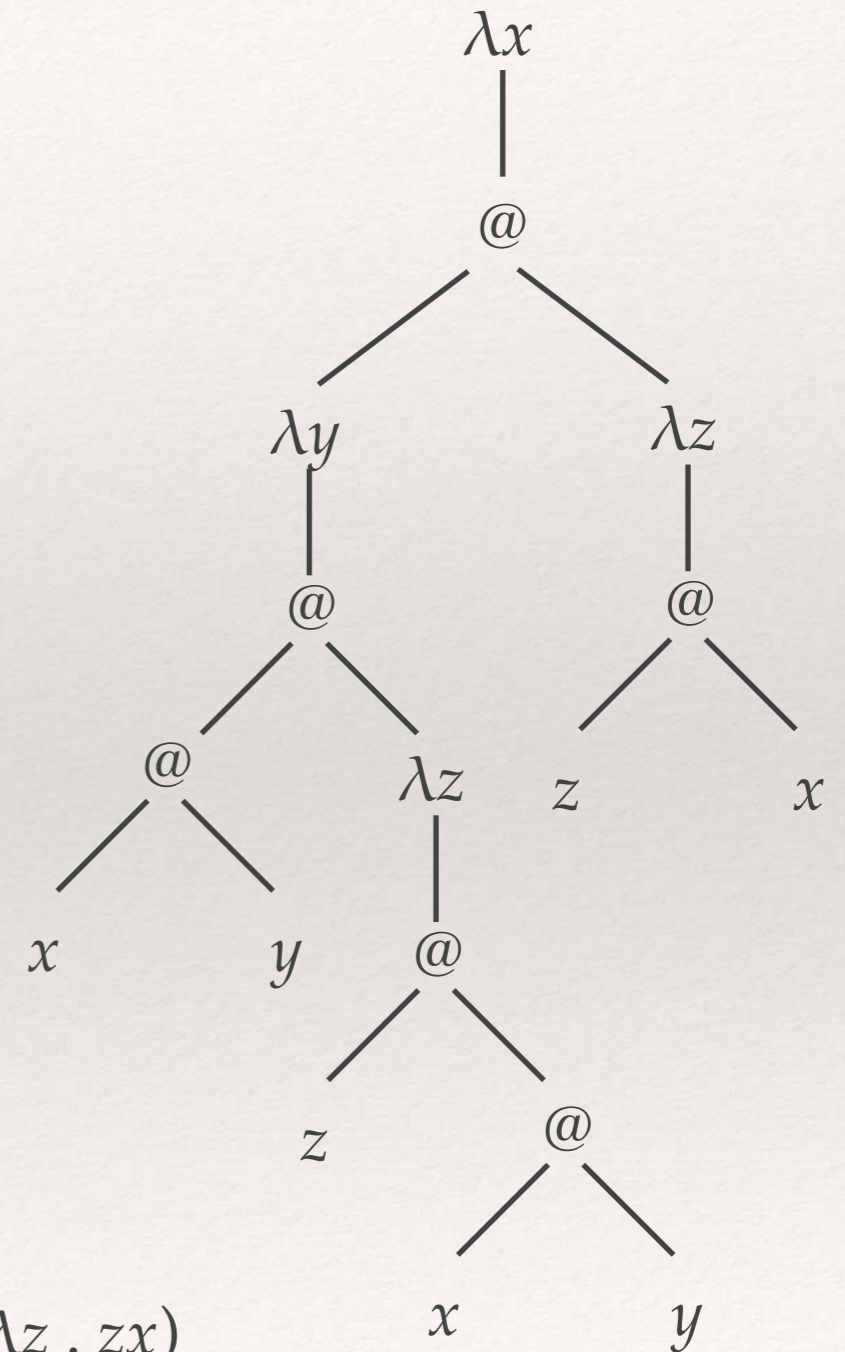
- ❖ Un terme est réellement un **arbre**.
La syntaxe représente ces arbres, modulo les conventions usuelles de **parenthésage** et de **priorités** (à la Caml)



$\lambda x . (\lambda y . xy(\lambda z . z(xy))) (\lambda z . zx)$

La syntaxe du λ -calcul

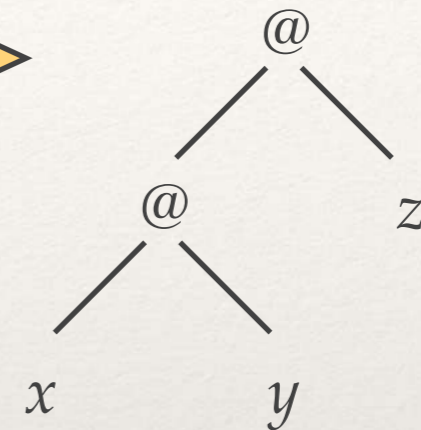
- ❖ Un terme est réellement un **arbre**.
La syntaxe représente ces arbres, modulo les conventions usuelles de **parenthésage** et de **priorités** (à la Caml)
- ❖ Quelques exemples...



$\lambda x . (\lambda y . xy(\lambda z . z(xy))) (\lambda z . zx)$

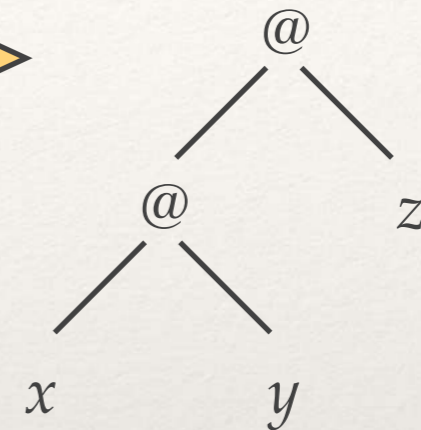
La syntaxe du λ -calcul

❖ xyz dénote $(xy)z$,



La syntaxe du λ -calcul

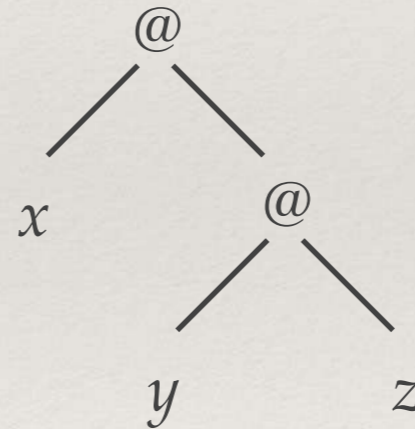
❖ xyz dénote $(xy)z$,



❖ pas $x(yz)$

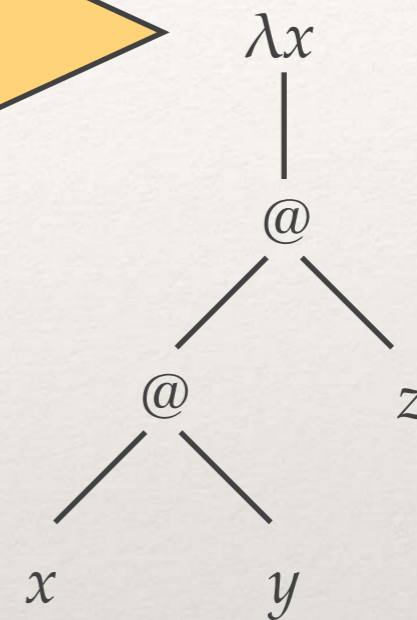


❖ L'application n'est **pas** associative



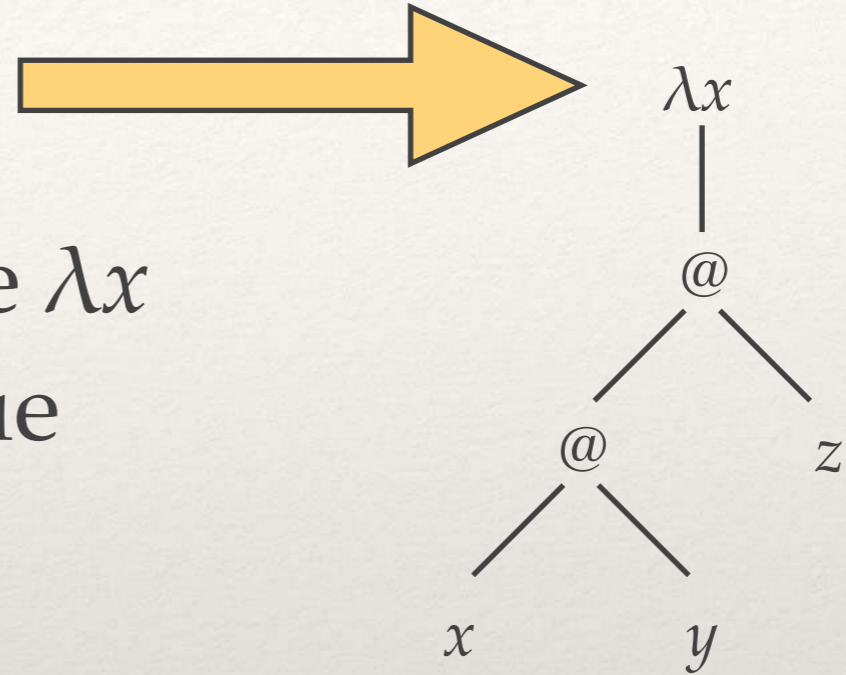
La syntaxe du λ -calcul

❖ $\lambda x . xyz$ dénote $(\lambda x . (xyz))$



La syntaxe du λ -calcul

- ❖ $\lambda x . xyz$ dénote $(\lambda x . (xyz))$
- ❖ ... autrement dit la portée de λx s'étend aussi loin à droite que possible

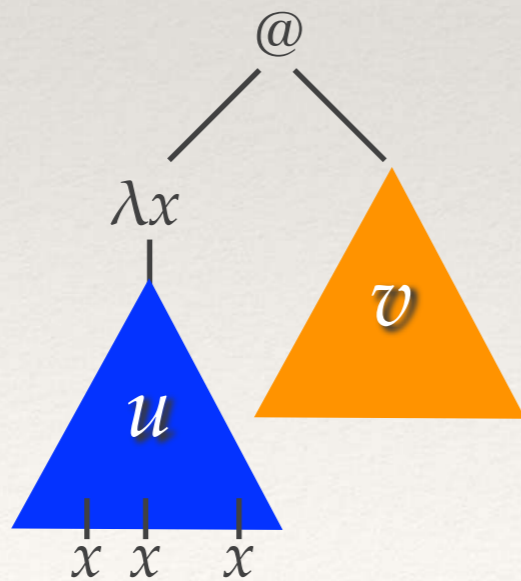


Calcul: la β -réduction

La β -réduction

❖ Une seule règle de calcul:

$$(\beta) \quad (\lambda x . u) v \rightarrow u[x:=v]$$

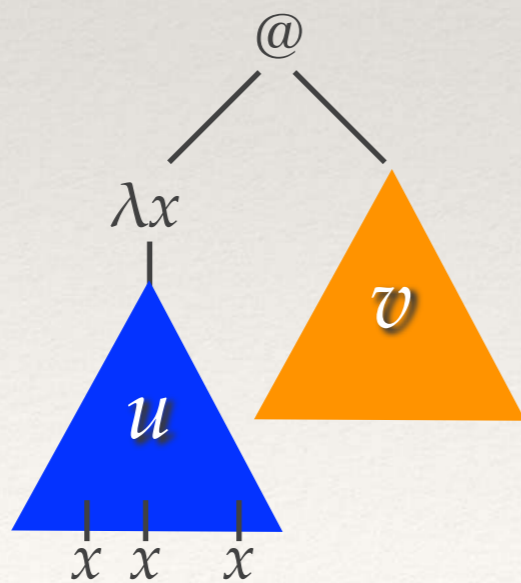


La β -réduction

rédex

❖ Une seule règle de calcul:

$$(\beta) \quad (\lambda x . u) v \rightarrow u[x:=v]$$



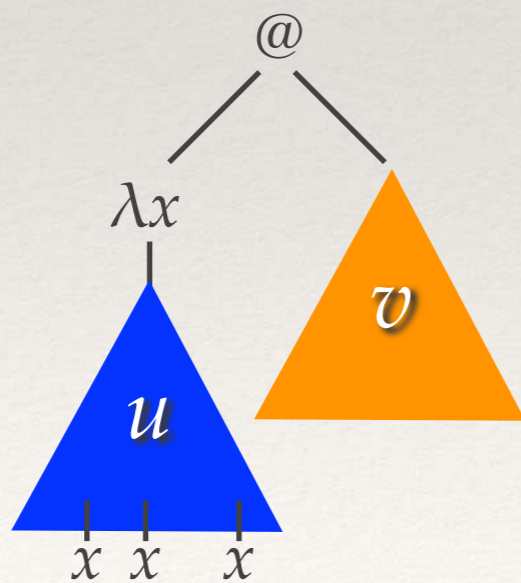
La β -réduction

rédex

contractum

❖ Une seule règle de calcul:

$$(\beta) \quad (\lambda x . u) v \rightarrow u[x:=v]$$



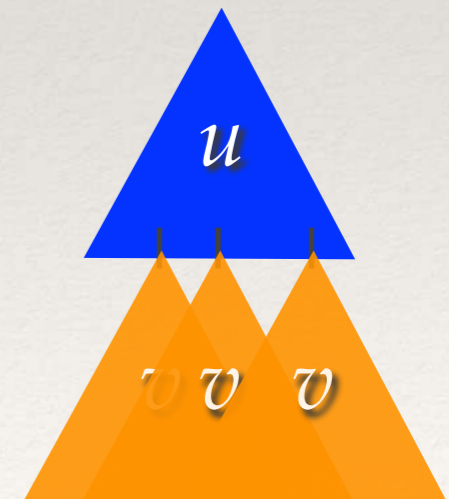
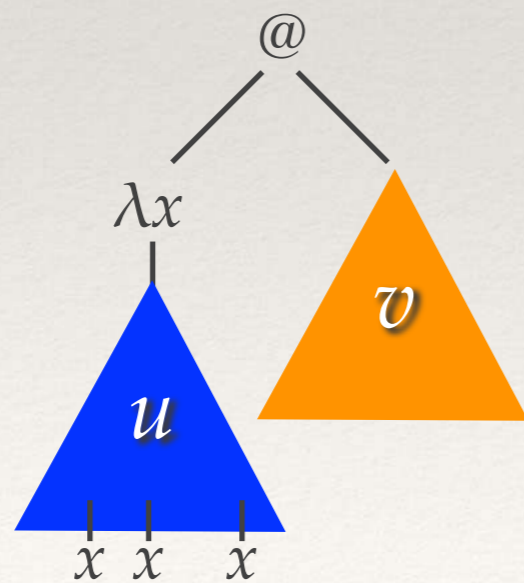
La β -réduction

rédex

contractum

❖ Une seule règle de calcul:

$$(\beta) \quad (\lambda x . u) v \rightarrow u[x:=v]$$



La β -réduction

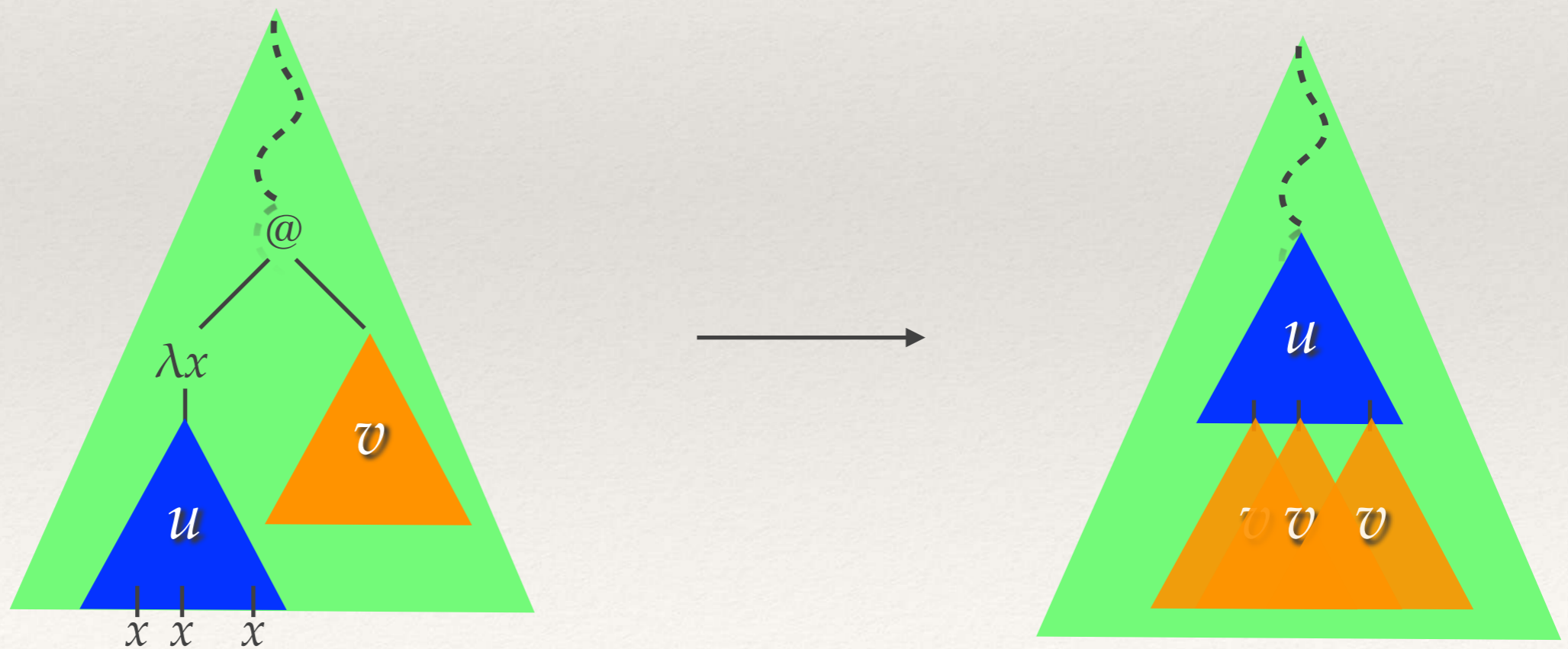
rédex

contractum

- ❖ Une seule règle de calcul:

$$(\beta) \quad (\lambda x . u) v \rightarrow u[x:=v]$$

- ❖ applicable n'importe où dans un terme



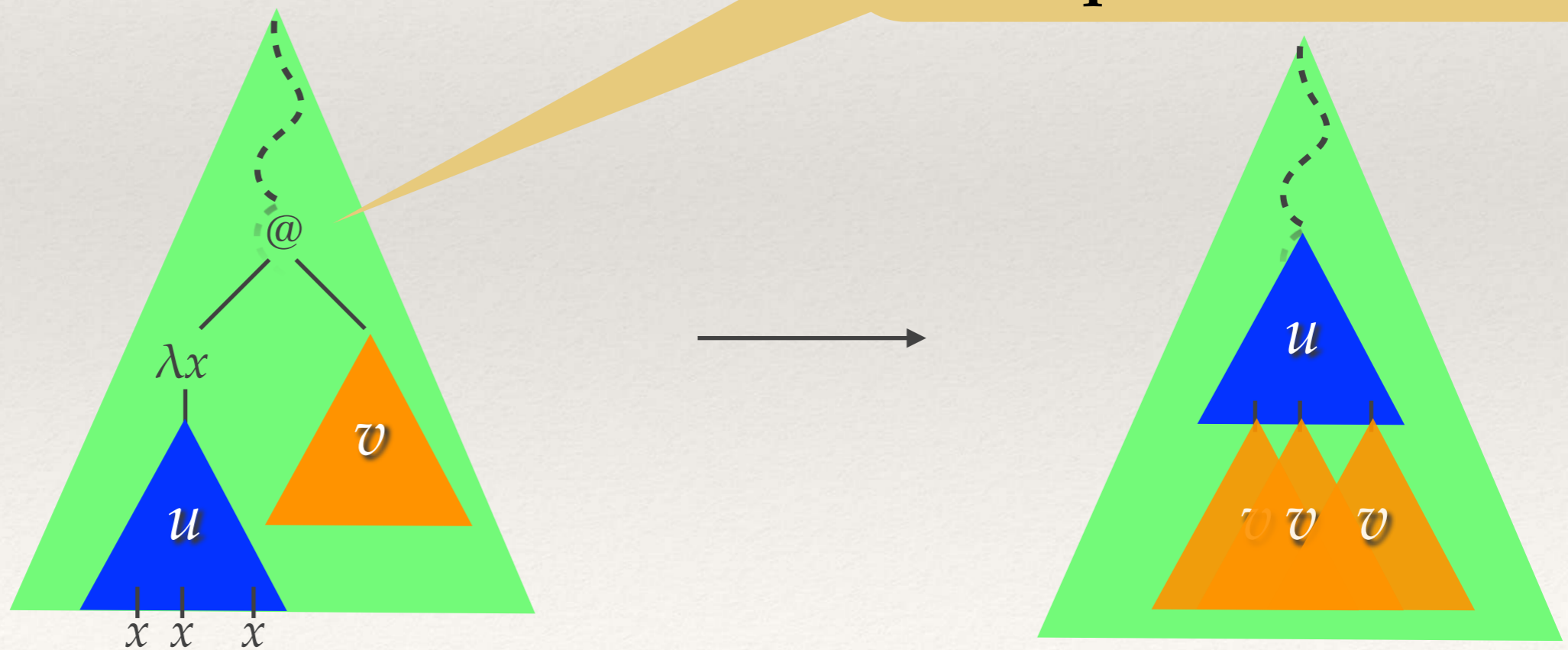
La β -réduction

rédex

contractum

- ❖ Une seule règle de calcul:
(β) $(\lambda x . u) v \rightarrow u[x:=v]$
- ❖ applicable n'importe où dans un terme

Il peut y avoir plusieurs rédexes dans un terme, mais on n'en contracte **qu'un** à la fois



Autres relations

- ❖ On écrira \rightarrow pour la β -réduction, mais parfois aussi pour n'importe quelle autre relation de réduction (relation binaire...)
- ❖ Si ambiguïté, on écrira \rightarrow_{β} pour la β -réduction

Autres relations

- ❖ On écrira \rightarrow pour la β -réduction, mais parfois aussi pour n'importe quelle autre relation de réduction (relation binaire...)
- ❖ Si ambiguïté, on écrira \rightarrow_{β} pour la β -réduction
- ❖ **Exemple:** on peut ajouter la règle
$$(\eta) \quad \lambda x . ux \rightarrow u \quad (\text{si } x \text{ pas libre dans } u)$$
et considérer la $\beta\eta$ -réduction $\rightarrow_{\beta\eta}$
(Non, on ne peut pas simuler (η) par (β) , cf. $\lambda x . yx$ où y est une variable $\neq x$)

Clôtures

- ❖ \rightarrow^* = clôture **réflexive-transitive** de \rightarrow (étoile de Kleene)
= plus petit préordre contenant \rightarrow
 $u \rightarrow^* v$ ssi il existe une **réduction** (un chemin)
 $u=u_0 \rightarrow u_1 \rightarrow \dots u_{n-1} \rightarrow u_n=v$, avec $n \geq 0$

Clôtures

- ❖ \rightarrow^* = clôture **réflexive-transitive** de \rightarrow (étoile de Kleene)
= plus petit préordre contenant \rightarrow
 $u \rightarrow^* v$ ssi il existe une **réduction** (un chemin)
 $u=u_0 \rightarrow u_1 \rightarrow \dots u_{n-1} \rightarrow u_n=v$, avec $n \geq 0$
- ❖ \rightarrow^+ = clôture **transitive** de \rightarrow
= plus petite relation transitive contenant \rightarrow
 $u \rightarrow^+ v$ ssi il existe une **réduction non vide**
 $u=u_0 \rightarrow u_1 \rightarrow \dots u_{n-1} \rightarrow u_n=v$, avec $n \geq 1$

Clôtures

- ❖ \rightarrow^* = clôture **réflexive-transitive** de \rightarrow (étoile de Kleene)
= plus petit préordre contenant \rightarrow
 $u \rightarrow^* v$ ssi il existe une **réduction** (un chemin)
 $u = u_0 \rightarrow u_1 \rightarrow \dots u_{n-1} \rightarrow u_n = v$, avec $n \geq 0$
- ❖ \rightarrow^+ = clôture **transitive** de \rightarrow
= plus petite relation transitive contenant \rightarrow
 $u \rightarrow^+ v$ ssi il existe une **réduction non vide**
 $u = u_0 \rightarrow u_1 \rightarrow \dots u_{n-1} \rightarrow u_n = v$, avec $n \geq 1$
- ❖ \leftrightarrow^* = $(\leftarrow \cup \rightarrow)^*$ est la **β -équivalence** (parfois notée $=_\beta$)

Substitution

Substitution?

- ❖ $(\beta) \quad (\lambda x . u) v \rightarrow u[x:=v]$
- ❖ D'accord, mais comment définit-on **formellement** la substitution $u[x:=v]$ de v pour x dans u ?
- ❖ Bizarrement, c'est une question très compliquée (et très enquiquillante — on s'empressera d'ignorer les difficultés à l'avenir)

Substitution: 1er essai

- ❖ Substitution textuelle: non, donnerait:

$$(\lambda x . u) [x:=v] = \lambda v . (u[x:=v])$$

n'est même pas syntaxiquement correct
(sauf si v est une variable)

Substitution: 2ème essai

- ❖ Définition par récurrence sur u :

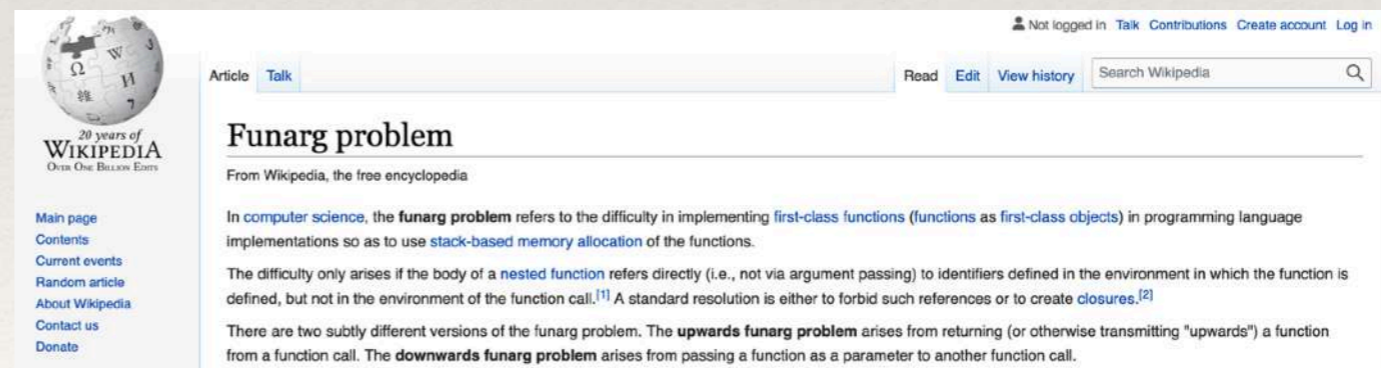
$$x [x:=v] \stackrel{\text{def}}{=} v$$

$$y [x:=v] \stackrel{\text{def}}{=} y \quad (y \neq x)$$

$$(st) [x:=v] \stackrel{\text{def}}{=} (s [x:=v]) (t [x:=v])$$

$$(\lambda z . u) [x:=v] \stackrel{\text{def}}{=} \lambda z . (u[x:=v])$$

- ❖ A l'air bien défini, mais souffre de quelques problèmes...



The screenshot shows a Wikipedia article page for "Funarg problem". The page includes a navigation bar at the top with links for "Not logged in", "Talk", "Contributions", "Create account", and "Log in". Below the navigation bar is a search box and a "Search Wikipedia" button. The article title "Funarg problem" is prominently displayed, followed by the text "From Wikipedia, the free encyclopedia". The main content of the article discusses the difficulty of implementing first-class functions in programming languages, specifically mentioning stack-based memory allocation and nested functions. It also notes that the difficulty arises if a nested function refers directly to identifiers defined in the environment of the function call. The article concludes by mentioning two versions of the funarg problem: the upwards funarg problem and the downwards funarg problem.

Le problème avec la substitution

- ❖ D'après la définition,
 $(\lambda x . x) [x:=y] = \lambda x . y$
 $(\lambda y . x) [x:=y] = \lambda y . y$

$$x [x:=v] \stackrel{\text{def}}{=} v$$

$$y [x:=v] \stackrel{\text{def}}{=} y \quad (y \neq x)$$

$$(st) [x:=v] \stackrel{\text{def}}{=} (s [x:=v]) (t [x:=v])$$

$$(\lambda z . u) [x:=v] \stackrel{\text{def}}{=} \lambda z . (u[x:=v])$$

Le problème avec la substitution

- ❖ D'après la définition,
 $(\lambda x . x) [x:=y] = \lambda x . y$
 $(\lambda y . x) [x:=y] = \lambda y . y$
- ❖ Pour corriger ça, on va:

$$x [x:=v] \stackrel{\text{def}}{=} v$$

$$y [x:=v] \stackrel{\text{def}}{=} y \quad (y \neq x)$$

$$(st) [x:=v] \stackrel{\text{def}}{=} (s [x:=v]) (t [x:=v])$$

$$(\lambda z . u) [x:=v] \stackrel{\text{def}}{=} \lambda z . (u[x:=v])$$

Le problème avec la substitution

- ❖ D'après la définition,
 $(\lambda x . x) [x:=y] = \lambda x . y$
 $(\lambda y . x) [x:=y] = \lambda y . y$

$$x [x:=v] \stackrel{\text{def}}{=} v$$

$$y [x:=v] \stackrel{\text{def}}{=} y \quad (y \neq x)$$

$$(st) [x:=v] \stackrel{\text{def}}{=} (s [x:=v]) (t [x:=v])$$

$$(\lambda z . u) [x:=v] \stackrel{\text{def}}{=} \lambda z . (u[x:=v])$$

- ❖ Pour corriger ça, on va:
- ❖ restreindre la substitution par des conditions sur les variables **libres** et **liées**

Le problème avec la substitution

- ❖ D'après la définition,
 $(\lambda x . x) [x:=y] = \lambda x . y$
 $(\lambda y . x) [x:=y] = \lambda y . y$

$$\begin{aligned}x [x:=v] &\stackrel{\text{def}}{=} v \\y [x:=v] &\stackrel{\text{def}}{=} y \quad (y \neq x) \\(st) [x:=v] &\stackrel{\text{def}}{=} (s [x:=v]) (t [x:=v]) \\(\lambda z . u) [x:=v] &\stackrel{\text{def}}{=} \lambda z . (u[x:=v])\end{aligned}$$

- ❖ Pour corriger ça, on va:
- ❖ restreindre la substitution par des conditions sur les variables **libres** et **liées**
- ❖ autoriser à **renommer** les variables liées (**α -renommage**)

Le problème avec la substitution

- ❖ D'après la définition,
 $(\lambda x . x) [x:=y] = \lambda x . y$
 $(\lambda y . x) [x:=y] = \lambda y . y$
- ❖ Pour corriger ça, on va:
- ❖ restreindre la substitution par des conditions sur les variables **libres** et **liées**
- ❖ autoriser à **renommer** les variables liées (**α -renommage**)
- ❖ autre solution: indices de **de Bruijn** (omis ici; voir exercices)

$$x [x:=v] \stackrel{\text{def}}{=} v$$

$$y [x:=v] \stackrel{\text{def}}{=} y \quad (y \neq x)$$

$$(st) [x:=v] \stackrel{\text{def}}{=} (s [x:=v]) (t [x:=v])$$

$$(\lambda z . u) [x:=v] \stackrel{\text{def}}{=} \lambda z . (u[x:=v])$$

Variables libres, variables liées

- ❖ $fv(u) = \{\text{variables libres de } u\}$
« qui apparaissent dans u à une position où on peut les remplacer »
- ❖ $bv(u) = \{\text{variables liées de } u\}$

$fv(x) \stackrel{\text{def}}{=} \{x\}$	$bv(x) \stackrel{\text{def}}{=} \emptyset$
$fv(uv) \stackrel{\text{def}}{=} fv(u) \cup fv(v)$	$bv(uv) \stackrel{\text{def}}{=} bv(u) \cup bv(v)$
$fv(\lambda x . u) \stackrel{\text{def}}{=} fv(u) - \{x\}$	$bv(\lambda x . u) \stackrel{\text{def}}{=} bv(u) \cup \{x\}$

Variables libres, variables liées

- ❖ $fv(u) = \{\text{variables libres de } u\}$
« qui apparaissent dans u à une position où on peut les remplacer »
- ❖ $bv(u) = \{\text{variables liées de } u\}$

$fv(x) \stackrel{\text{def}}{=} \{x\}$	$bv(x) \stackrel{\text{def}}{=} \emptyset$
$fv(uv) \stackrel{\text{def}}{=} fv(u) \cup fv(v)$	$bv(uv) \stackrel{\text{def}}{=} bv(u) \cup bv(v)$
$fv(\lambda x . u) \stackrel{\text{def}}{=} fv(u) - \{x\}$	$bv(\lambda x . u) \stackrel{\text{def}}{=} bv(u) \cup \{x\}$

- ❖ Exemples.

u	$fv(u)$	$bv(u)$
$\lambda x . x$?	?
$x(\lambda y . z)$?	?
$x(\lambda x . x)$?	?
$(\lambda x . xx)(y(\lambda z . yz)x)$?	?

Variables libres, variables liées

- ❖ $fv(u) = \{\text{variables libres de } u\}$
« qui apparaissent dans u à une position où on peut les remplacer »
- ❖ $bv(u) = \{\text{variables liées de } u\}$

$fv(x) \stackrel{\text{def}}{=} \{x\}$	$bv(x) \stackrel{\text{def}}{=} \emptyset$
$fv(uv) \stackrel{\text{def}}{=} fv(u) \cup fv(v)$	$bv(uv) \stackrel{\text{def}}{=} bv(u) \cup bv(v)$
$fv(\lambda x . u) \stackrel{\text{def}}{=} fv(u) - \{x\}$	$bv(\lambda x . u) \stackrel{\text{def}}{=} bv(u) \cup \{x\}$

- ❖ Exemples.

u	$fv(u)$	$bv(u)$
$\lambda x . x$	\emptyset	$\{x\}$
$x(\lambda y . z)$?	?
$x(\lambda x . x)$?	?
$(\lambda x . xx)(y(\lambda z . yz)x)$?	?

Variables libres, variables liées

- ❖ $fv(u) = \{\text{variables libres de } u\}$
« qui apparaissent dans u à une position où on peut les remplacer »
- ❖ $bv(u) = \{\text{variables liées de } u\}$

$fv(x) \stackrel{\text{def}}{=} \{x\}$	$bv(x) \stackrel{\text{def}}{=} \emptyset$
$fv(uv) \stackrel{\text{def}}{=} fv(u) \cup fv(v)$	$bv(uv) \stackrel{\text{def}}{=} bv(u) \cup bv(v)$
$fv(\lambda x . u) \stackrel{\text{def}}{=} fv(u) - \{x\}$	$bv(\lambda x . u) \stackrel{\text{def}}{=} bv(u) \cup \{x\}$

- ❖ Exemples.

u	$fv(u)$	$bv(u)$
$\lambda x . x$	\emptyset	$\{x\}$
$x(\lambda y . z)$	$\{x, z\}$	$\{y\}$
$x(\lambda x . x)$?	?
$(\lambda x . xx)(y(\lambda z . yz)x)$?	?

Variables libres, variables liées

- ❖ $fv(u) = \{\text{variables libres de } u\}$
« qui apparaissent dans u à une position où on peut les remplacer »
- ❖ $bv(u) = \{\text{variables liées de } u\}$

$fv(x) \stackrel{\text{def}}{=} \{x\}$	$bv(x) \stackrel{\text{def}}{=} \emptyset$
$fv(uv) \stackrel{\text{def}}{=} fv(u) \cup fv(v)$	$bv(uv) \stackrel{\text{def}}{=} bv(u) \cup bv(v)$
$fv(\lambda x . u) \stackrel{\text{def}}{=} fv(u) - \{x\}$	$bv(\lambda x . u) \stackrel{\text{def}}{=} bv(u) \cup \{x\}$

- ❖ Exemples.

u	$fv(u)$	$bv(u)$
$\lambda x . x$	\emptyset	$\{x\}$
$x(\lambda y . z)$	$\{x, z\}$	$\{y\}$
$x(\lambda x . x)$	$\{x\}$	$\{x\}$
$(\lambda x . xx)(y(\lambda z . yz)x)$?	?

oui, une variable peut être libre **et** liée!

Variables libres, variables liées

- ❖ $fv(u) = \{\text{variables libres de } u\}$
« qui apparaissent dans u à une position où on peut les remplacer »
- ❖ $bv(u) = \{\text{variables liées de } u\}$

$fv(x) \stackrel{\text{def}}{=} \{x\}$	$bv(x) \stackrel{\text{def}}{=} \emptyset$
$fv(uv) \stackrel{\text{def}}{=} fv(u) \cup fv(v)$	$bv(uv) \stackrel{\text{def}}{=} bv(u) \cup bv(v)$
$fv(\lambda x . u) \stackrel{\text{def}}{=} fv(u) - \{x\}$	$bv(\lambda x . u) \stackrel{\text{def}}{=} bv(u) \cup \{x\}$

- ❖ Exemples.

u	$fv(u)$	$bv(u)$
$\lambda x . x$	\emptyset	$\{x\}$
$x(\lambda y . z)$	$\{x, z\}$	$\{y\}$
$x(\lambda x . x)$	$\{x\}$	$\{x\}$
$(\lambda x . xx)(y(\lambda z . yz)x)$	$\{x, y\}$	$\{x, z\}$

oui, une variable peut être libre **et** liée!

Variables libres, variables liées

- ❖ $fv(u) = \{\text{variables libres de } u\}$
« qui apparaissent dans u à une position où on peut les remplacer »
- ❖ $bv(u) = \{\text{variables liées de } u\}$

$fv(x) \stackrel{\text{def}}{=} \{x\}$	$bv(x) \stackrel{\text{def}}{=} \emptyset$
$fv(uv) \stackrel{\text{def}}{=} fv(u) \cup fv(v)$	$bv(uv) \stackrel{\text{def}}{=} bv(u) \cup bv(v)$
$fv(\lambda x . u) \stackrel{\text{def}}{=} fv(u) - \{x\}$	$bv(\lambda x . u) \stackrel{\text{def}}{=} bv(u) \cup \{x\}$

- ❖ Exemples.

u	$fv(u)$	$bv(u)$
$\lambda x . x$	\emptyset	$\{x\}$
$x(\lambda y . z)$	$\{x, z\}$	$\{y\}$
$x(\lambda x . x)$	$\{x\}$	$\{x\}$
$(\lambda x . xx)(y(\lambda z . yz)x)$	$\{x, y\}$	$\{x, z\}$

oui, une variable peut être libre **et** liée!

au fait, la variable z' ($\neq x, y, z$) est-elle libre ici?

Substitution, 3ème essai

- ❖ On définit une opération de substitution **partielle**

$$u [x:=v]$$

définie uniquement lorsque

$$x \notin \text{bv}(u) \text{ et } \text{fv}(v) \cap \text{bv}(u) = \emptyset$$

$$x [x:=v] \stackrel{\text{def}}{=} v$$

$$y [x:=v] \stackrel{\text{def}}{=} y \quad (y \neq x)$$

$$(st) [x:=v] \stackrel{\text{def}}{=} (s [x:=v]) (t [x:=v])$$

$$(\lambda z . u) [x:=v] \stackrel{\text{def}}{=} \lambda z . (u[x:=v])$$

on dit que x est **substituable** par v dans u

Substitution, 3ème essai

- ❖ On définit une opération de substitution **partielle**

$$u [x:=v]$$

définie uniquement lorsque

$$x \notin \text{bv}(u) \text{ et } \text{fv}(v) \cap \text{bv}(u) = \emptyset$$

pour éviter le problème
« $(\lambda x . x) [x:=y] = \lambda x . y$ »

$$x [x:=v] \stackrel{\text{def}}{=} v$$

$$y [x:=v] \stackrel{\text{def}}{=} y \quad (y \neq x)$$

$$(st) [x:=v] \stackrel{\text{def}}{=} (s [x:=v]) (t [x:=v])$$

$$(\lambda z . u) [x:=v] \stackrel{\text{def}}{=} \lambda z . (u[x:=v])$$

on dit que x est
substituable
par v dans u

Substitution, 3ème essai

- ❖ On définit une opération de substitution **partielle**

$$u [x:=v]$$

définie uniquement lorsque

$$x \notin \text{bv}(u) \text{ et } \text{fv}(v) \cap \text{bv}(u) = \emptyset$$

on dit que x est **substituable** par v dans u

pour éviter le problème
« $(\lambda x . x) [x:=y] = \lambda x . y$ »

pour éviter le problème
« $(\lambda y . x) [x:=y] = \lambda y . y$ »

$$x [x:=v] \stackrel{\text{def}}{=} v$$

$$y [x:=v] \stackrel{\text{def}}{=} y \quad (y \neq x)$$

$$(st) [x:=v] \stackrel{\text{def}}{=} (s [x:=v]) (t [x:=v])$$

$$(\lambda z . u) [x:=v] \stackrel{\text{def}}{=} \lambda z . (u[x:=v])$$

Substitution, 3ème essai

- ❖ On définit une opération de substitution **partielle**

$$u [x:=v]$$

définie uniquement lorsque

$$x \notin \text{bv}(u) \text{ et } \text{fv}(v) \cap \text{bv}(u) = \emptyset$$

on dit que x est **substituable** par v dans u

pour éviter le problème
« $(\lambda x . x) [x:=y] = \lambda x . y$ »

pour éviter le problème
« $(\lambda y . x) [x:=y] = \lambda y . y$ »

- ❖ On peut assurer cette condition en **renommant** x ...

$$x [x:=v] \stackrel{\text{def}}{=} v$$

$$y [x:=v] \stackrel{\text{def}}{=} y \quad (y \neq x)$$

$$(st) [x:=v] \stackrel{\text{def}}{=} (s [x:=v]) (t [x:=v])$$

$$(\lambda z . u) [x:=v] \stackrel{\text{def}}{=} \lambda z . (u[x:=v])$$

α -renommage

- ❖ On souhaite considérer que
« $\lambda x . u(x)$ et $\lambda y . u(y)$ sont interchangeables »
- ❖ On définit la relation α par:
$$\lambda x . u \alpha \lambda y . (u[x:=y])$$

à condition que x soit substituable par y dans u
et que y ne soit pas libre dans u
(sinon on aurait $\lambda x . xy \alpha \lambda y . yy$)

i.e., $x, y \notin \text{bv}(u)$
et $y \notin \text{fv}(u)$
- ❖ La relation d' **α -équivalence** $=_{\alpha}$ est la plus petite congruence
(rel. d'équivalence compatible aux contextes) contenant α

Propriétés du α -renommage

$$\lambda x . u \ \alpha \ \lambda y . (u[x:=y])$$
$$(x, y \notin \text{bv}(u), y \notin \text{fv}(u))$$

❖ Exemple: $\lambda x . \lambda y . xy =_{\alpha} \lambda x . \lambda z . xz$

(avec $z \neq x, y$ —noter le passage au contexte)

$$=_{\alpha} \lambda y . \lambda z . yz =_{\alpha} \lambda y . \lambda x . yx$$

Propriétés du α -renommage

$$\lambda x . u \ \alpha \ \lambda y . (u[x:=y])$$
$$(x, y \notin \text{bv}(u), y \notin \text{fv}(u))$$

❖ Exemple: $\lambda x . \lambda y . xy =_{\alpha} \lambda x . \lambda z . xz$

(avec $z \neq x, y$ — noter le passage au contexte)

$$=_{\alpha} \lambda y . \lambda z . yz =_{\alpha} \lambda y . \lambda x . yx$$

❖ On peut toujours α -renommer $\lambda x . u$

de sorte que $u[x:=v]$ soit bien défini

(= de sorte que $x \notin \text{bv}(u)$ et $\text{fv}(v) \cap \text{bv}(u) = \emptyset$)

Propriétés du α -renommage

$$\lambda x . u \ \alpha \ \lambda y . (u[x:=y])$$
$$(x, y \notin \text{bv}(u), y \notin \text{fv}(u))$$

❖ Exemple: $\lambda x . \lambda y . xy =_{\alpha} \lambda x . \lambda z . xz$

(avec $z \neq x, y$ — noter le passage au contexte)

$$=_{\alpha} \lambda y . \lambda z . yz =_{\alpha} \lambda y . \lambda x . yx$$

❖ On peut toujours α -renommer $\lambda x . u$

de sorte que $u[x:=v]$ soit bien défini

(= de sorte que $x \notin \text{bv}(u)$ et $\text{fv}(v) \cap \text{bv}(u) = \emptyset$)

❖ En pratique, on utilisera la **convention de Barendregt**:

renommer systématiquement toutes les variables liées pour que:

— aucune variable n'est liée 2 fois

— aucune variable n'est liée et libre

Propriétés du α -renommage

$$\lambda x . u \ \alpha \ \lambda y . (u[x:=y])$$
$$(x, y \notin \text{bv}(u), y \notin \text{fv}(u))$$

❖ Exemple: $\lambda x . \lambda y . xy =_{\alpha} \lambda x . \lambda z . xz$

(avec $z \neq x, y$ — noter le passage au contexte)

$$=_{\alpha} \lambda y . \lambda z . yz =_{\alpha} \lambda y . \lambda x . yx$$

❖ On peut toujours α -renommer $\lambda x . u$

de sorte que $u[x:=v]$ soit bien défini

(= de sorte que $x \notin \text{bv}(u)$ et $\text{fv}(v) \cap \text{bv}(u) = \emptyset$)

❖ En pratique, on utilisera la **convention de Barendregt**:

renommer systématiquement toutes les variables liées pour que:

— aucune variable n'est liée 2 fois

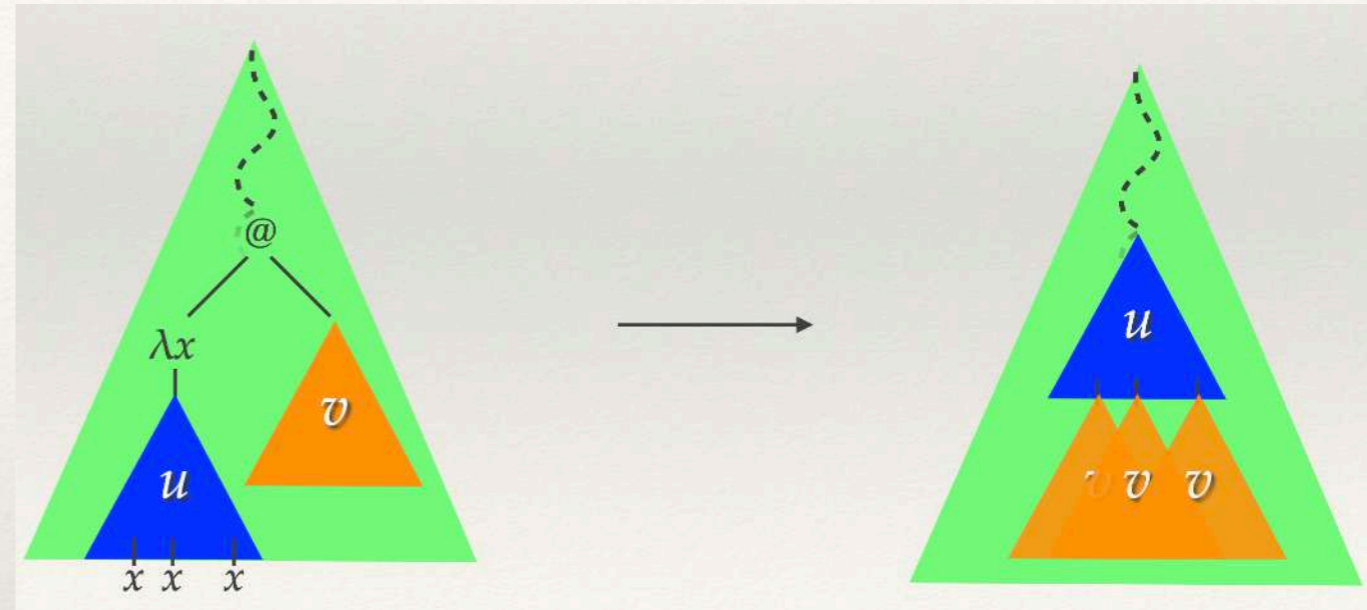
— aucune variable n'est liée et libre

❖ On peut maintenant (re)définir proprement la β -réduction.

La β -réduction... vraiment

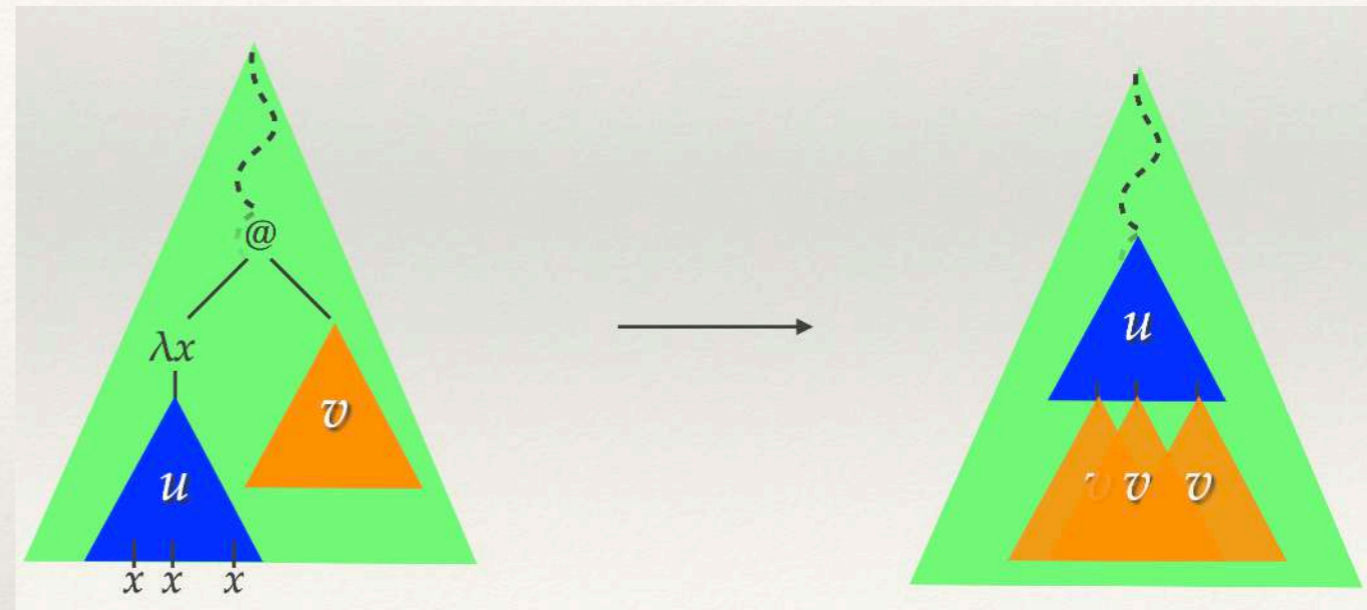
La β -réduction, vraiment

❖ $(\beta) \quad (\lambda x . u) v \rightarrow u[x:=v]$



La β -réduction, vraiment

- ❖ $(\beta) \quad (\lambda x . u) v \rightarrow u[x:=v]$
- ❖ On dit que $s \rightarrow t$ ssi il existe un **contexte** C et un **redex** $(\lambda x . u) v$ tels que $s =_{\alpha} C[(\lambda x . u) v]$ et $t =_{\alpha} C[u[x:=v]]$

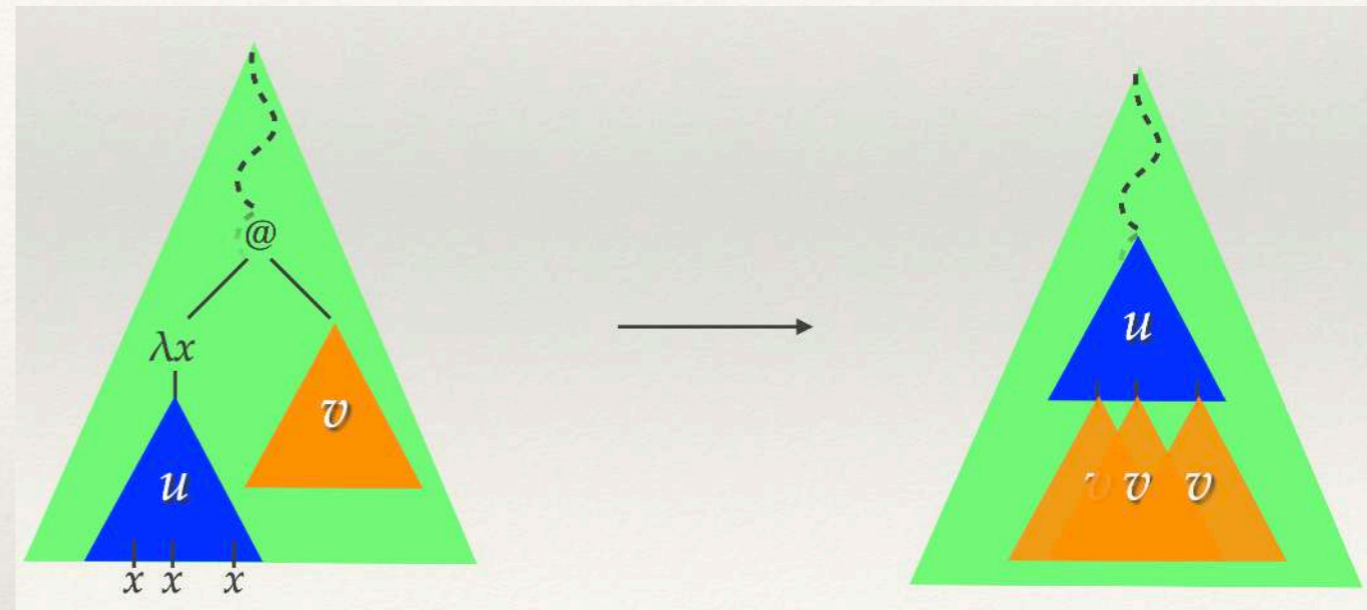


La β -réduction, vraiment

❖ $(\beta) \quad (\lambda x . u) v \rightarrow u[x:=v]$

❖ On dit que $s \rightarrow t$ ssi
il existe un **contexte** C
et un redex $(\lambda x . u) v$
tels que $s =_{\alpha} C[(\lambda x . u) v]$
et $t =_{\alpha} C[u[x:=v]]$

❖ $C ::= _ \quad$ trou (où le terme est inséré)
| $\lambda x . C$ réduction « sous la lambda »
| $C v$ la réduction s'opère dans la fonction
| $u C$ la réduction s'opère dans l'argument



Un exemple de réductions

$$(\lambda y . (\lambda x . yxx)(\lambda z . z(xy)))(\lambda z' . z')$$

Un exemple de réductions

$(\lambda y . \underline{(\lambda x . yxx)(\lambda z . z(xy))})(\lambda z' . z')$

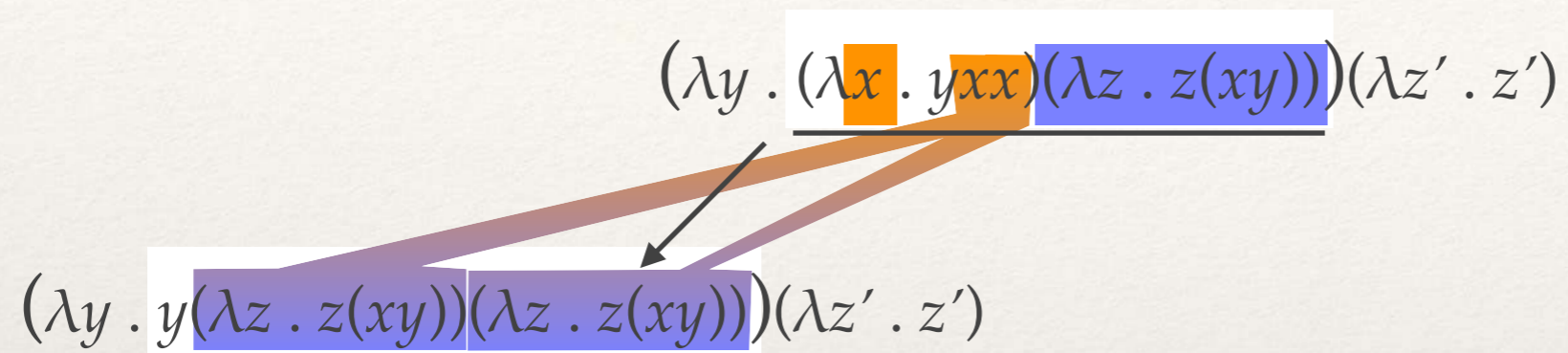
Un exemple de réductions

$(\lambda y . (\lambda x . yxx)(\lambda z . z(xy)))(\lambda z' . z')$

Un exemple de réductions

$(\lambda y . (\lambda x . yxx)(\lambda z . z(xy)))(\lambda z' . z')$

Un exemple de réductions





Un exemple de réductions



$$(\lambda y . (\lambda x . yxx)(\lambda z . z(xy)))(\lambda z' . z')$$

$$\underline{(\lambda y . y(\lambda z . z(xy))(\lambda z . z(xy)))(\lambda z' . z')}$$

Un exemple de réductions

$$(\lambda y . (\lambda x . yxx)(\lambda z . z(xy)))(\lambda z' . z')$$

$$\underline{(\lambda y . y(\lambda z . z(xy))(\lambda z . z(xy)))(\lambda z' . z')}$$

$$(\lambda z' . z')(\lambda z . z(x(\lambda z' . z')))(\lambda z . z(x(\lambda z' . z')))$$

Un exemple de réductions

$$(\lambda y . (\lambda x . yxx)(\lambda z . z(xy)))(\lambda z' . z')$$

$$\underline{(\lambda y . y(\lambda z . z(xy))(\lambda z . z(xy)))(\lambda z' . z')}$$

$$\underline{(\lambda z' . z')(\lambda z . z(x(\lambda z' . z')))(\lambda z . z(x(\lambda z' . z')))}$$

Un exemple de réductions

$(\lambda y . (\lambda x . yxx)(\lambda z . z(xy)))(\lambda z' . z')$

$(\lambda y . y(\lambda z . z(xy))(\lambda z . z(xy)))(\lambda z' . z')$

$(\lambda z' . z')(\lambda z . z(x(\lambda z' . z')))(\lambda z . z(x(\lambda z' . z')))$

$(\lambda z . z(x(\lambda z' . z')))(\lambda z . z(x(\lambda z' . z')))$

Un exemple de réductions

$(\lambda y . (\lambda x . yxx)(\lambda z . z(xy)))(\lambda z' . z')$

$(\lambda y . y(\lambda z . z(xy))(\lambda z . z(xy)))(\lambda z' . z')$

$(\lambda z' . z')(\lambda z . z(x(\lambda z' . z')))(\lambda z . z(x(\lambda z' . z')))$

$(\lambda z . z(x(\lambda z' . z')))(\lambda z . z(x(\lambda z' . z')))$

Un exemple de réductions

$$(\lambda y . (\lambda x . yxx)(\lambda z . z(xy)))(\lambda z' . z')$$
$$(\lambda y . y(\lambda z . z(xy))(\lambda z . z(xy)))(\lambda z' . z')$$
$$(\lambda z' . z')(\lambda z . z(x(\lambda z' . z')))(\lambda z . z(x(\lambda z' . z')))$$
$$(\lambda z . z(x(\lambda z' . z')))(\lambda z . z(x(\lambda z' . z')))$$
$$(\lambda z . z(x(\lambda z' . z')))(x(\lambda z' . z'))$$

Un exemple de réductions

$(\lambda y . (\lambda x . yxx)(\lambda z . z(xy)))(\lambda z' . z')$

$(\lambda y . y(\lambda z . z(xy))(\lambda z . z(xy)))(\lambda z' . z')$

$(\lambda z' . z')(\lambda z . z(x(\lambda z' . z')))(\lambda z . z(x(\lambda z' . z')))$

$(\lambda z . z(x(\lambda z' . z')))(\lambda z . z(x(\lambda z' . z')))$

$(\lambda z . z(x(\lambda z' . z')))(x(\lambda z' . z'))$

Un exemple de réductions

$(\lambda y . (\lambda x . yxx)(\lambda z . z(xy)))(\lambda z' . z')$

$(\lambda y . y(\lambda z . z(xy))(\lambda z . z(xy)))(\lambda z' . z')$

$(\lambda z' . z')(\lambda z . z(x(\lambda z' . z')))(\lambda z . z(x(\lambda z' . z')))$

$(\lambda z . z(x(\lambda z' . z')))(\lambda z . z(x(\lambda z' . z')))$

$(\lambda z . z(x(\lambda z' . z')))(x(\lambda z' . z'))$

$x(\lambda z' . z')(x(\lambda z' . z'))$

Un exemple de réductions

$(\lambda y . (\lambda x . yxx)(\lambda z . z(xy)))(\lambda z' . z')$

$(\lambda y . y(\lambda z . z(xy))(\lambda z . z(xy)))(\lambda z' . z')$

$(\lambda z' . z')(\lambda z . z(x(\lambda z' . z')))(\lambda z . z(x(\lambda z' . z')))$

$(\lambda z . z(x(\lambda z' . z')))(\lambda z . z(x(\lambda z' . z')))$

$(\lambda z . z(x(\lambda z' . z')))(x(\lambda z' . z'))$

$x(\lambda z' . z')(x(\lambda z' . z'))$

Forme normale
(=sans redex)

Un exemple de réductions

$(\lambda y . (\lambda x . yxx)(\lambda z . z(xy)))(\lambda z' . z')$

2 rédexes

$(\lambda y . y(\lambda z . z(xy))(\lambda z . z(xy)))(\lambda z' . z')$

$(\lambda z' . z')(\lambda z . z(x(\lambda z' . z')))(\lambda z . z(x(\lambda z' . z')))$

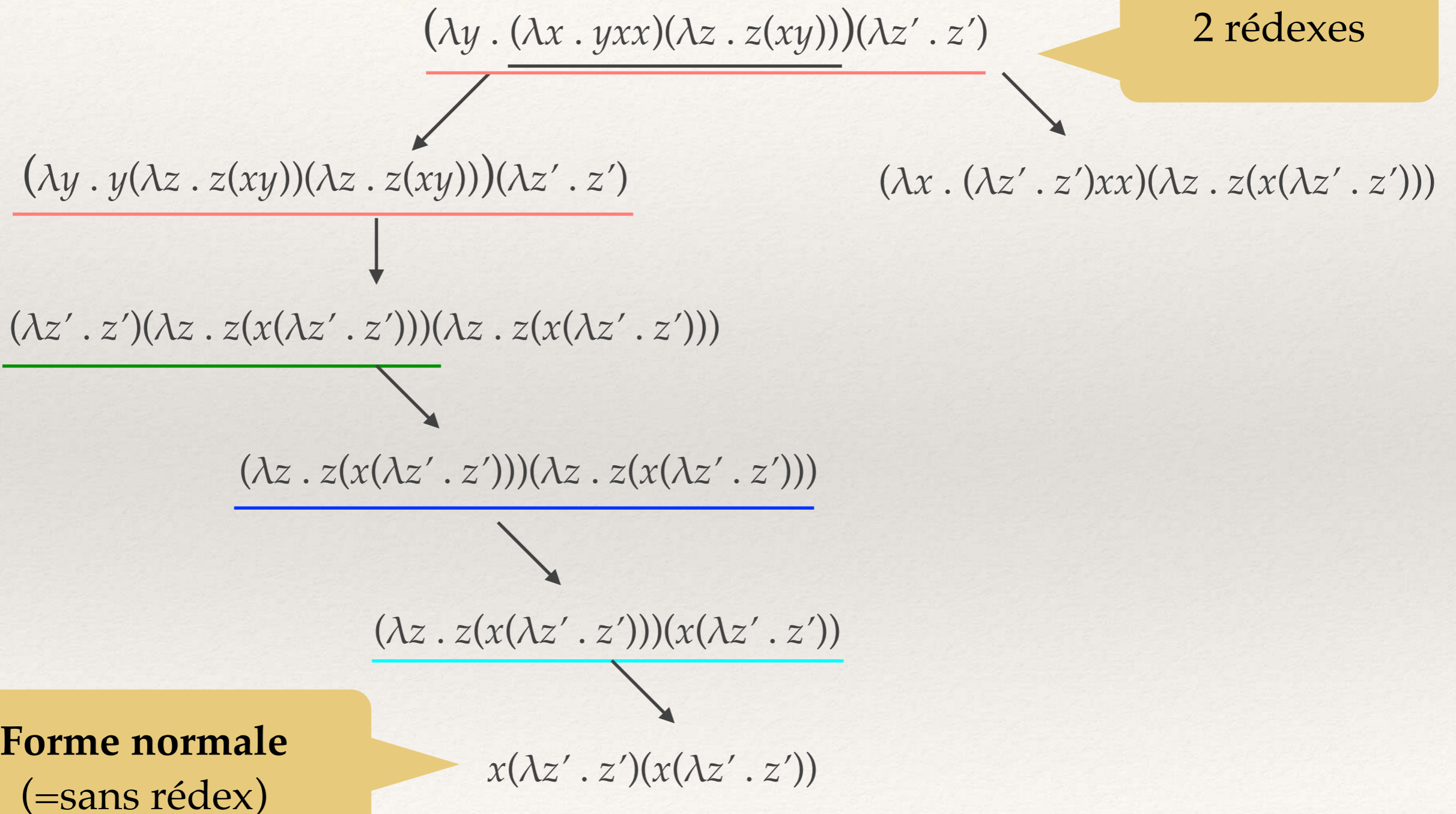
$(\lambda z . z(x(\lambda z' . z')))(\lambda z . z(x(\lambda z' . z')))$

$(\lambda z . z(x(\lambda z' . z')))(x(\lambda z' . z'))$

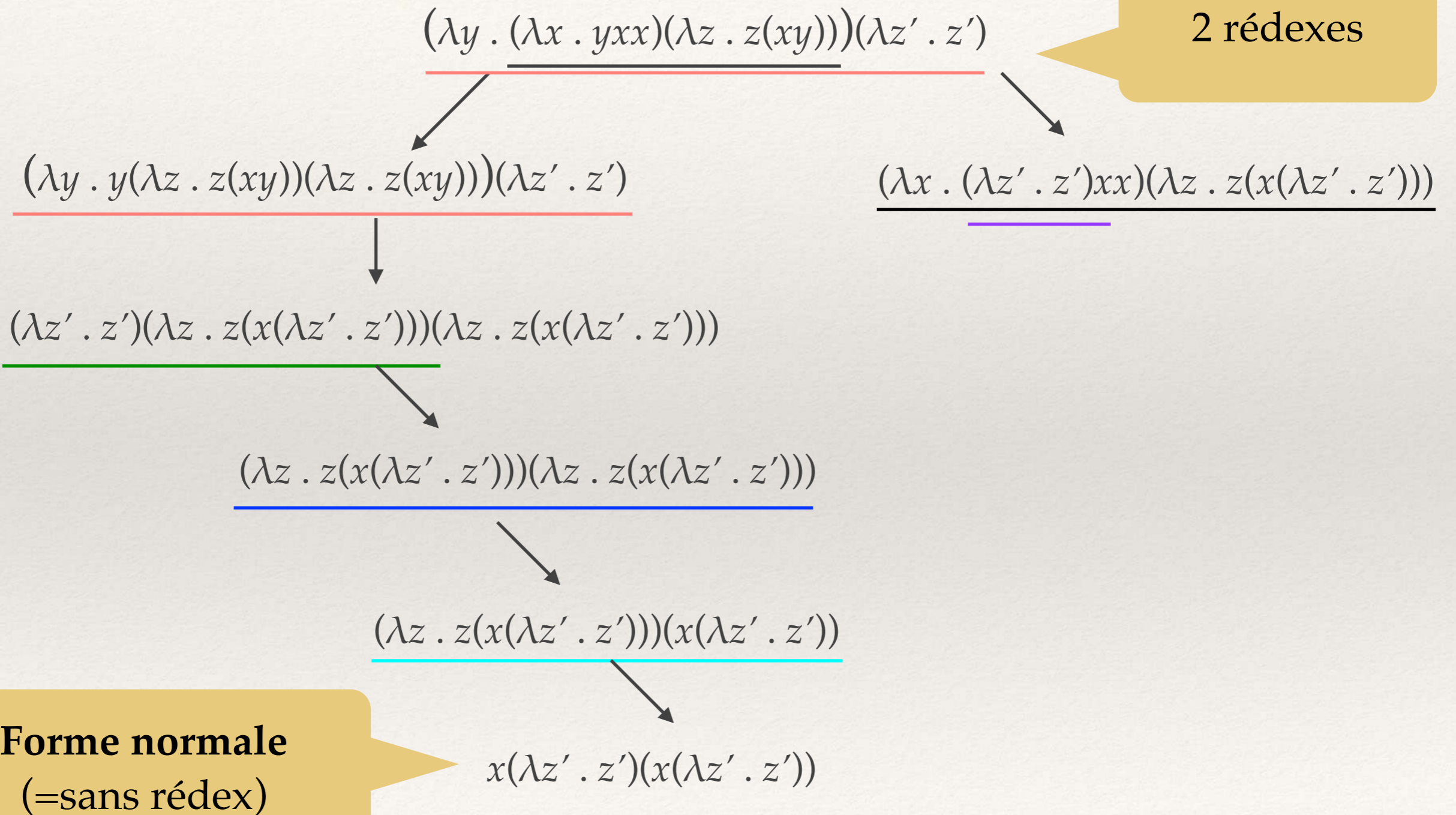
$x(\lambda z' . z')(x(\lambda z' . z'))$

Forme normale
(=sans rédex)

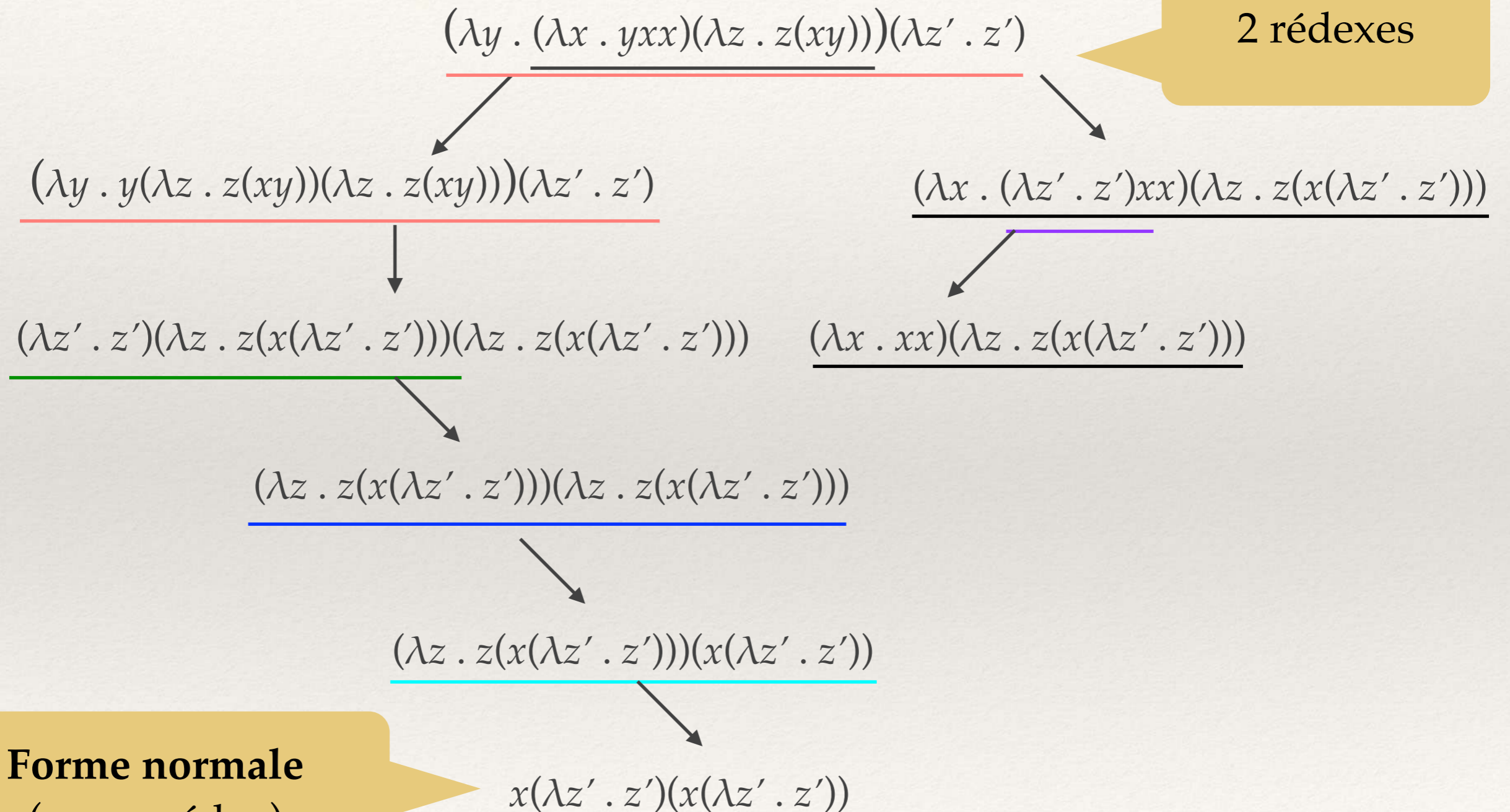
Un exemple de réductions



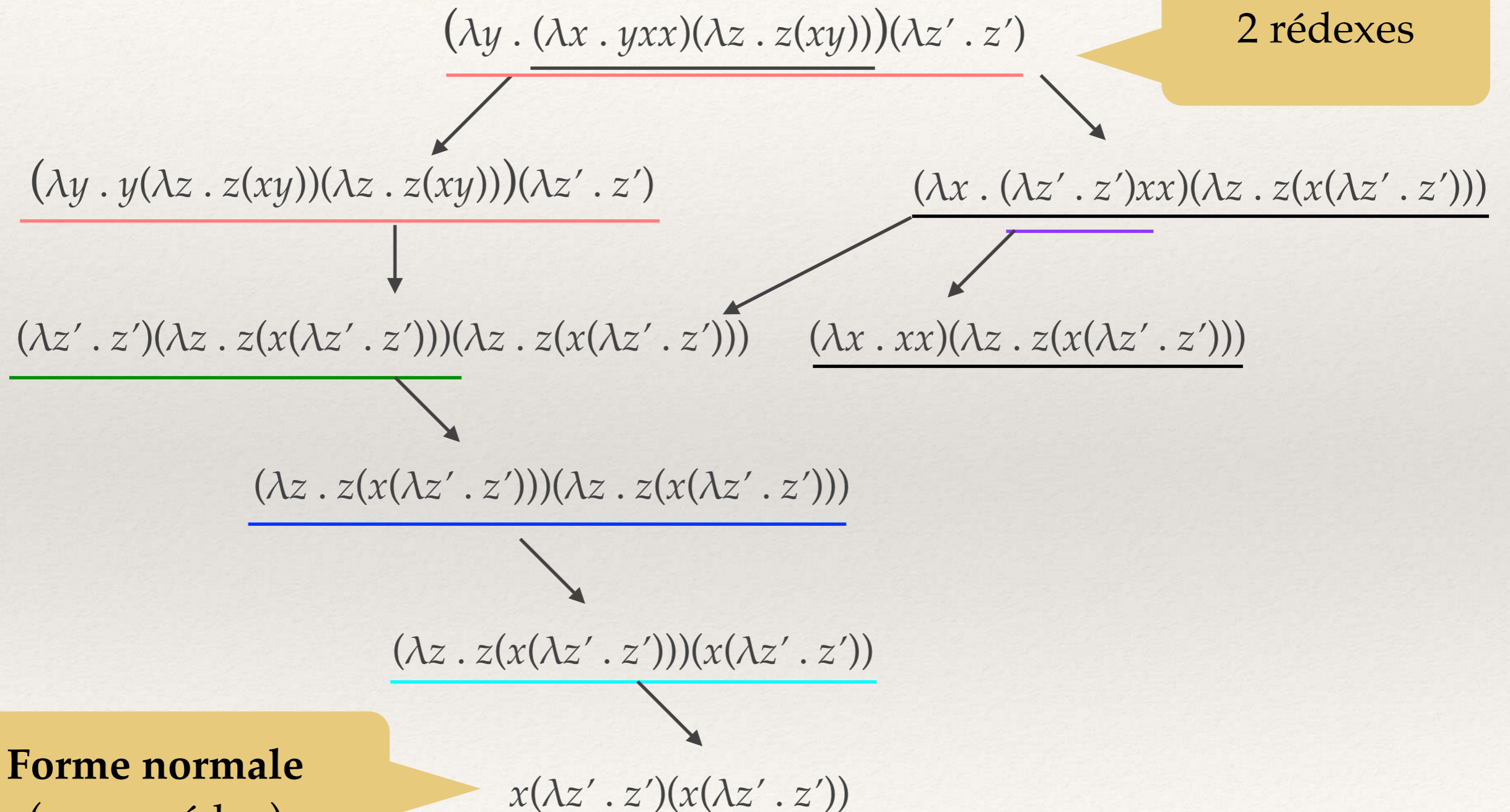
Un exemple de réductions



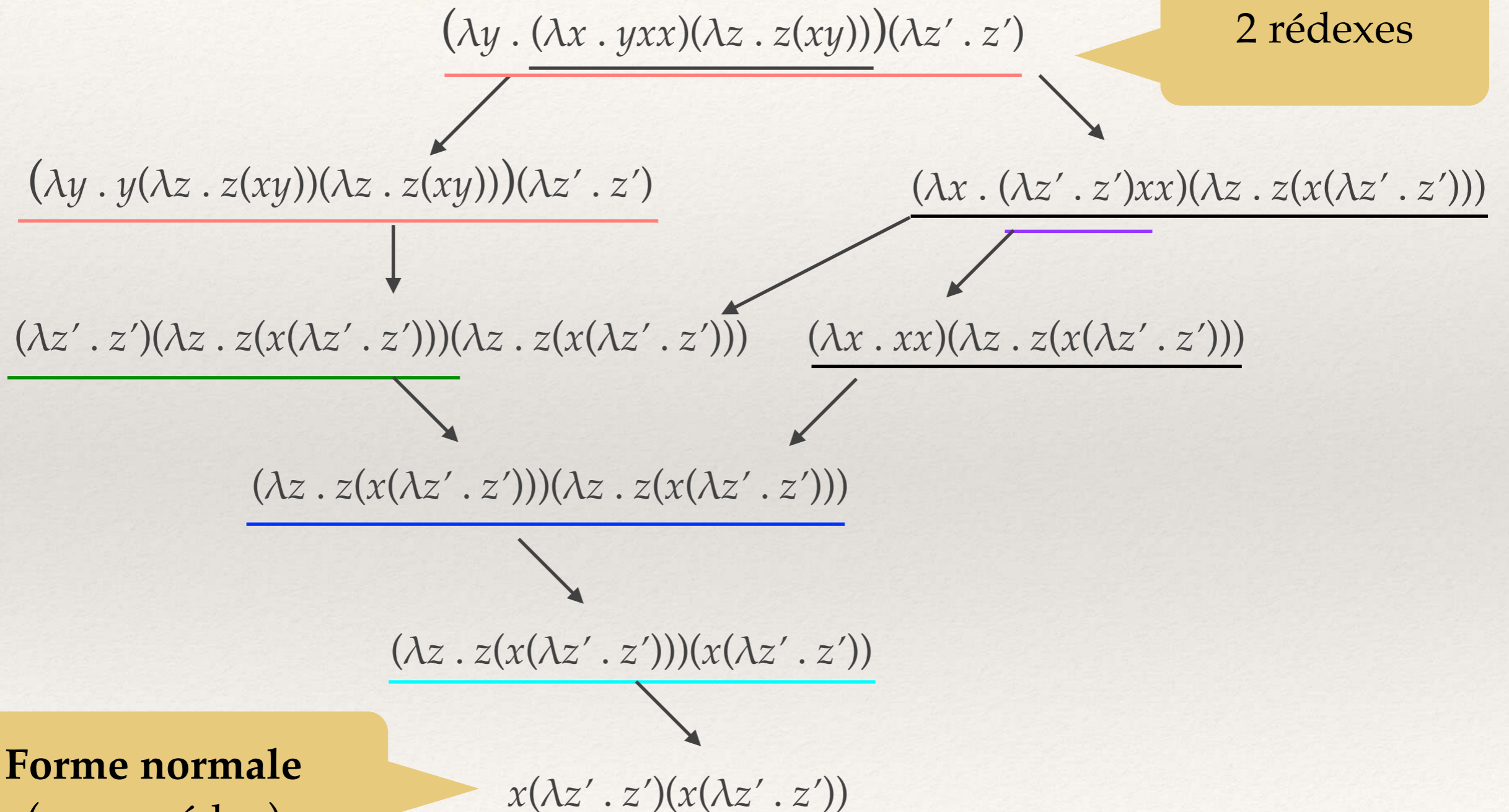
Un exemple de réductions



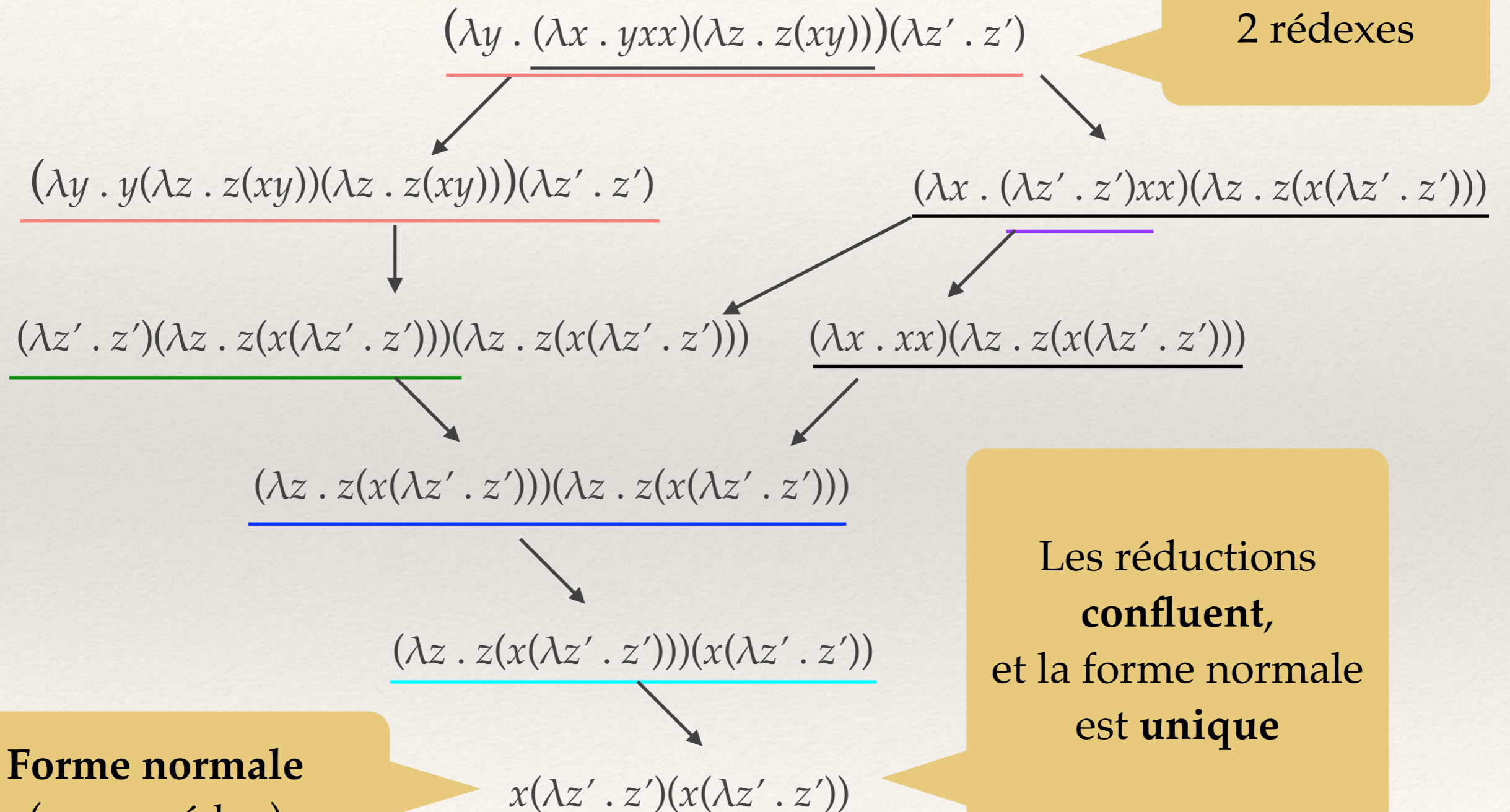
Un exemple de réductions



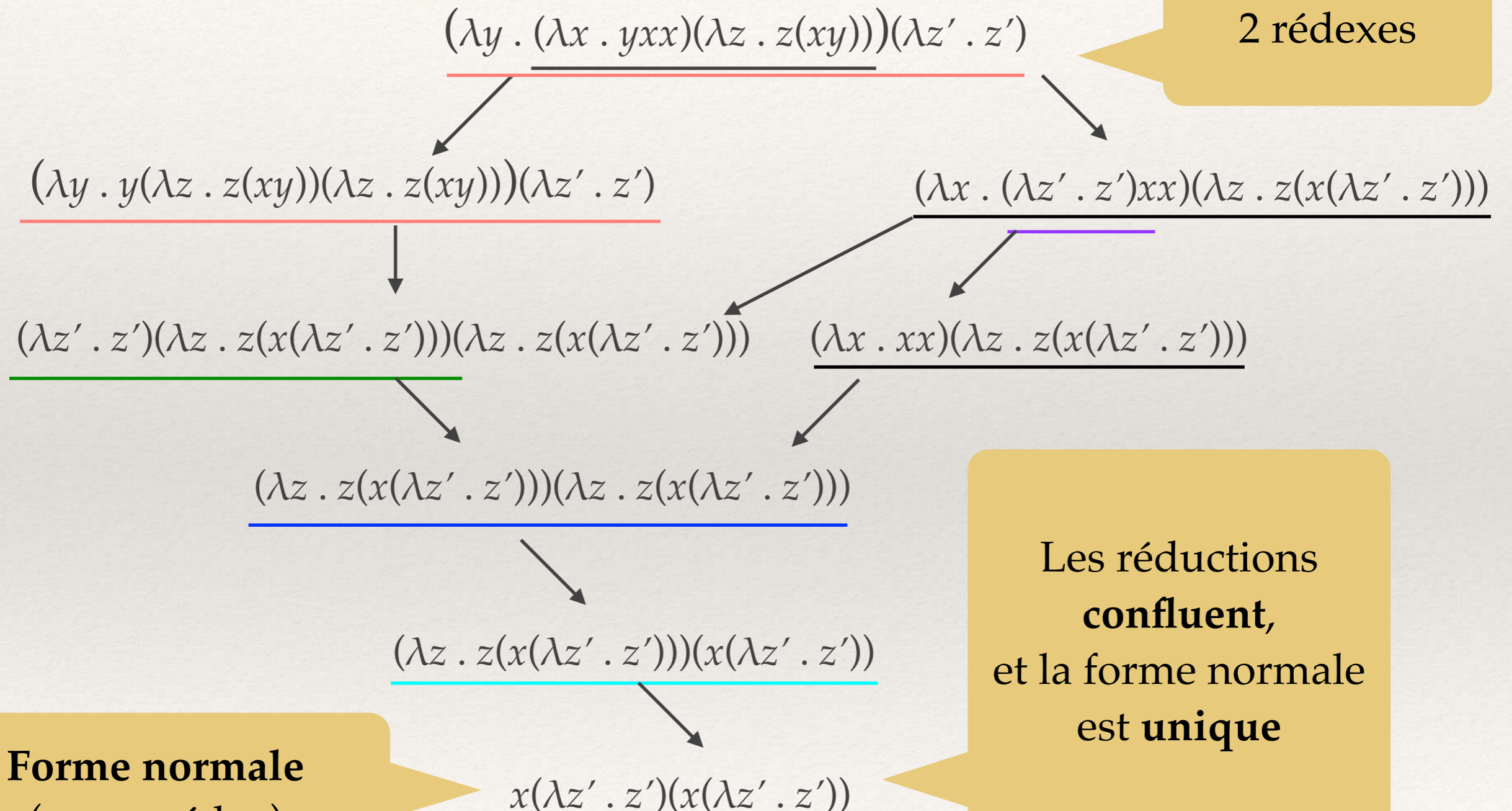
Un exemple de réductions



Un exemple de réductions



Un exemple de réductions



2 redexes

Forme normale
(=sans redex)

Les réductions
confluent,
et la forme normale
est **unique**

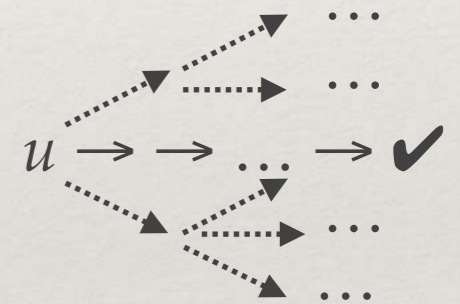
Sont-ce des phénomènes généraux?

Terminaison

- ❖ Une **forme normale** est un terme u sans rédex,
i.e., $u \nrightarrow$

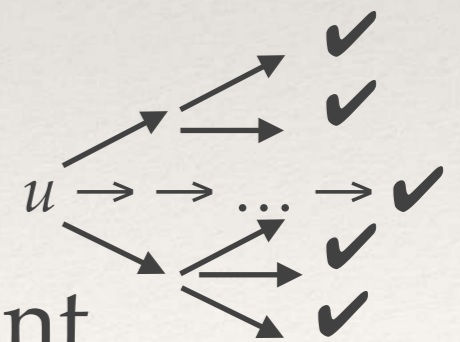
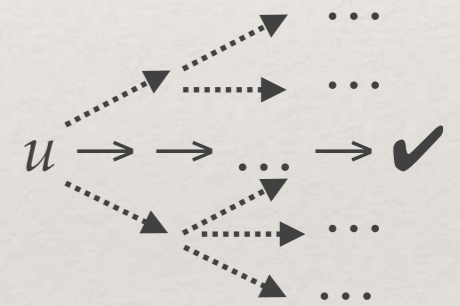
Terminaison

- ❖ Une **forme normale** est un terme u sans rédex,
i.e., $u \not\rightarrow$
- ❖ Un terme u est **normalisable** (= **faiblement terminant**)
ssi il a une forme normale
ssi **il existe** une réduction partant
de u qui termine

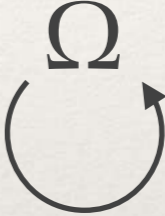


Terminaison

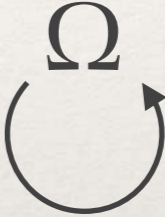
- ❖ Une **forme normale** est un terme u sans rédex,
i.e., $u \not\rightarrow$
- ❖ Un terme u est **normalisable** (= faiblement terminant)
ssi il a une forme normale
ssi **il existe** une réduction partant
de u qui termine
- ❖ Un terme u est **fortement normalisable** (=terminant)
ssi **toutes** les réductions partant de u terminent



Le terme Ω

- ❖ Posons $\Omega \stackrel{\text{def}}{=} \delta\delta$, où $\delta \stackrel{\text{def}}{=} \lambda x . xx$ [auto-application]
- ❖ Son « arbre » de réductions est:
 $(\Omega = (\lambda x . xx) \delta \rightarrow \delta\delta = \Omega,$
et c'est la seule réduction possible)

- ❖ Ω est un terme qui n'est (même) **pas normalisable**

Le terme Ω

- ❖ Posons $\Omega \stackrel{\text{def}}{=} \delta\delta$, où $\delta \stackrel{\text{def}}{=} \lambda x . xx$ [auto-application]
- ❖ Son « arbre » de réductions est:
 $(\Omega = (\lambda x . xx) \delta \rightarrow \delta\delta = \Omega,$
et c'est la seule réduction possible)

- ❖ Ω est un terme qui n'est (même) **pas normalisable**
- ❖ Y a-t-il un terme normalisable mais pas fortement?

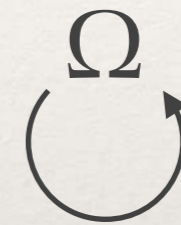
Le terme Ω

❖ Posons $\Omega \stackrel{\text{def}}{=} \delta\delta$, où $\delta \stackrel{\text{def}}{=} \lambda x . xx$ [auto-application]

❖ Son « arbre » de réductions est:

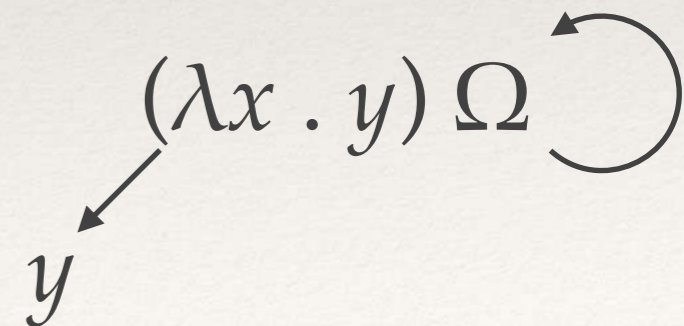
$$(\Omega = (\lambda x . xx) \delta \rightarrow \delta\delta = \Omega,$$

et c'est la seule réduction possible)



❖ Ω est un terme qui n'est (même) **pas normalisable**

❖ Y a-t-il un terme normalisable mais pas fortement?



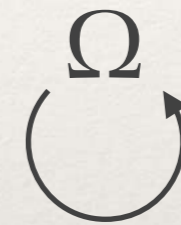
Le terme Ω

❖ Posons $\Omega \stackrel{\text{def}}{=} \delta\delta$, où $\delta \stackrel{\text{def}}{=} \lambda x . xx$ [auto-application]

❖ Son « arbre » de réductions est:

$$(\Omega = (\lambda x . xx) \delta \rightarrow \delta\delta = \Omega,$$

et c'est la seule réduction possible)

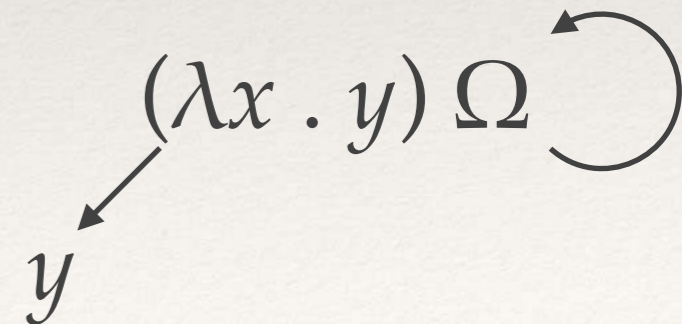


❖ Ω est un terme qui n'est (même) **pas normalisable**

❖ Y a-t-il un terme normalisable mais pas fortement?

❖ On a donc tous les cas possibles:

fortement norm. / norm. / pas normalisable



Confluence

confluence forte

confluence

Church-Rosser

confluence locale

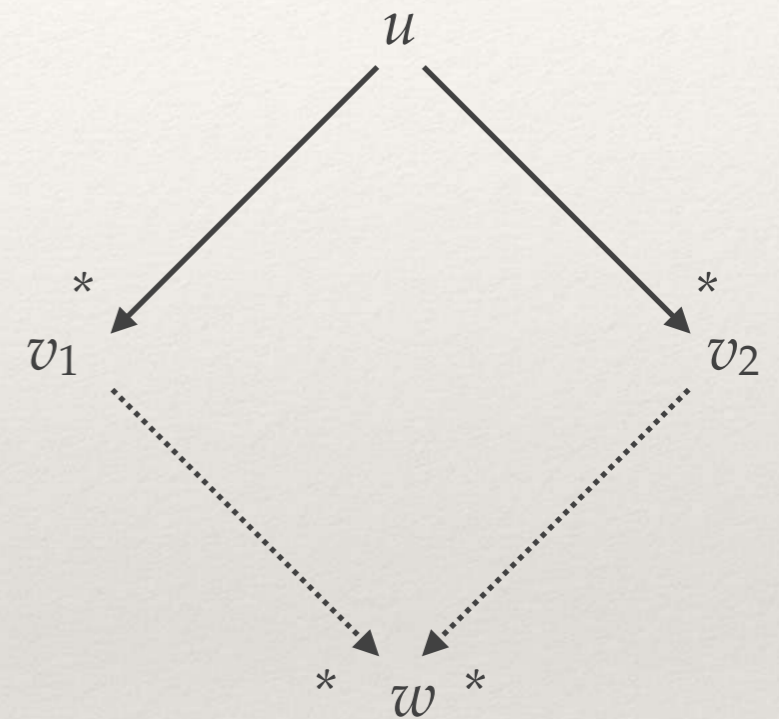
forme normale unique

Propriété de forme normale unique

- ❖ On dit qu'une relation de réduction a la **propriété de forme normale unique** ssi tout terme a **au plus une** forme normale
- ❖ On verra que la β -réduction a cette propriété.

Confluence

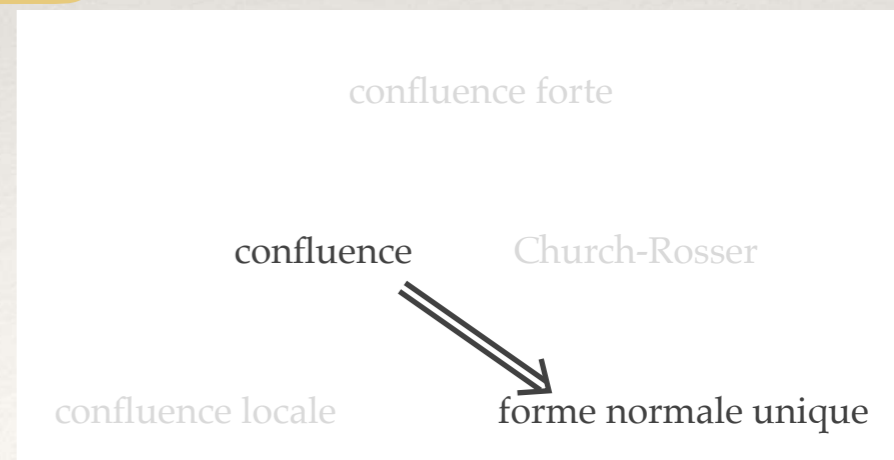
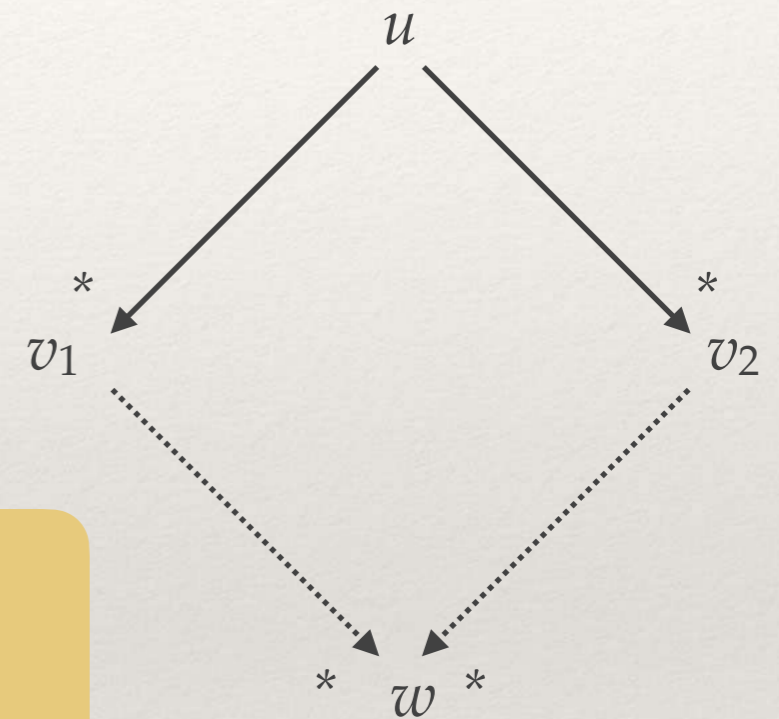
- ❖ Une relation de réduction \rightarrow est **confluente** ssi toutes les réductions $u \rightarrow^* v_1$ et $u \rightarrow^* v_2$ sont **joignables**, i.e. il existe des réductions $v_1 \rightarrow^* w$ et $v_2 \rightarrow^* w$ vers le même terme w



	confluence forte	
confluence		Church-Rosser
confluence locale		forme normale unique

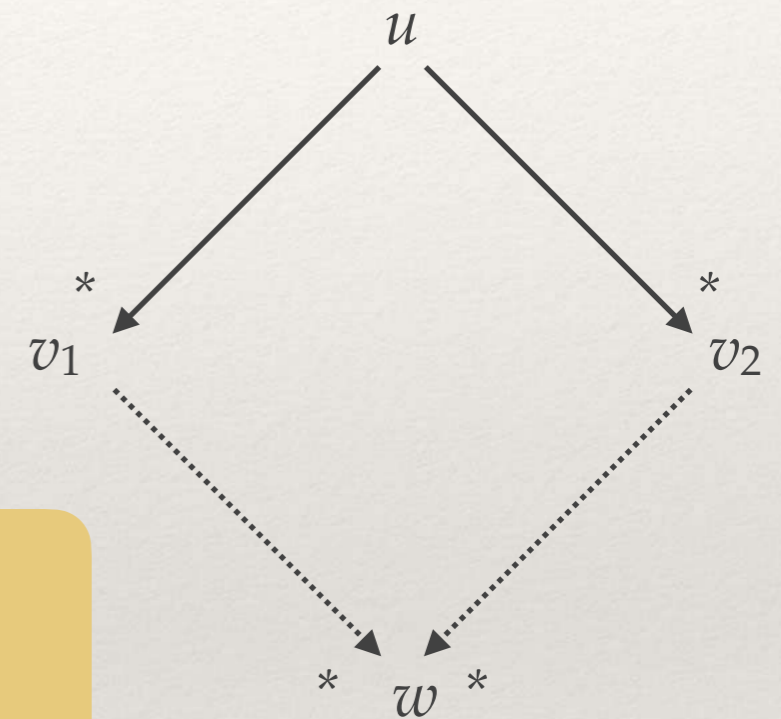
Confluence

- ❖ Une relation de réduction \rightarrow est **confluente** ssi toutes les réductions $u \rightarrow^* v_1$ et $u \rightarrow^* v_2$ sont **joignables**, i.e. il existe des réductions $v_1 \rightarrow^* w$ et $v_2 \rightarrow^* w$ vers le même terme w
- ❖ **Fait.** Toute relation confluente a la propriété de forme normale unique.



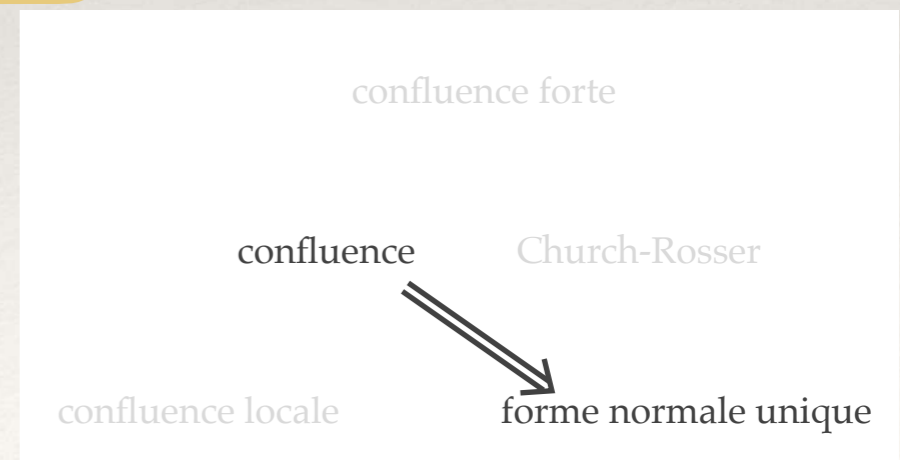
Confluence

- ❖ Une relation de réduction \rightarrow est **confluente** ssi toutes les réductions $u \rightarrow^* v_1$ et $u \rightarrow^* v_2$ sont **joignables**, i.e. il existe des réductions $v_1 \rightarrow^* w$ et $v_2 \rightarrow^* w$ vers le même terme w



- ❖ **Fait.** Toute relation confluente a la propriété de forme normale unique.

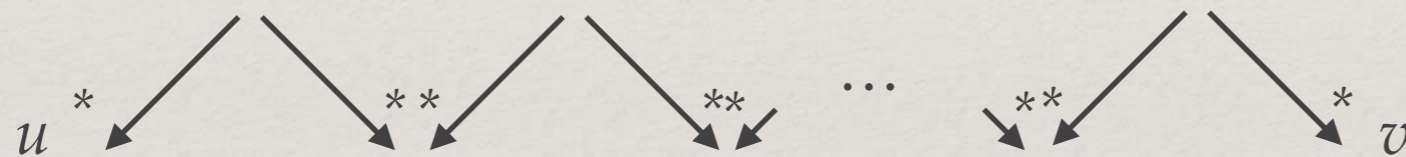
- ❖ *Preuve.* Si v_1 et v_2 sont deux formes normales de u , les réductions $v_1 \rightarrow^* w$ et $v_2 \rightarrow^* w$ sont de longueur 0.



Propriété de Church-Rosser

- ❖ \rightarrow a la propriété de **Church-Rosser** ssi $\leftrightarrow^* = \rightarrow^*; \leftarrow^*$
i.e., pour tous u, v tels que $u \leftrightarrow^* v$, il existe w tel que
$$u \rightarrow^* w \text{ et } w \leftarrow^* v$$

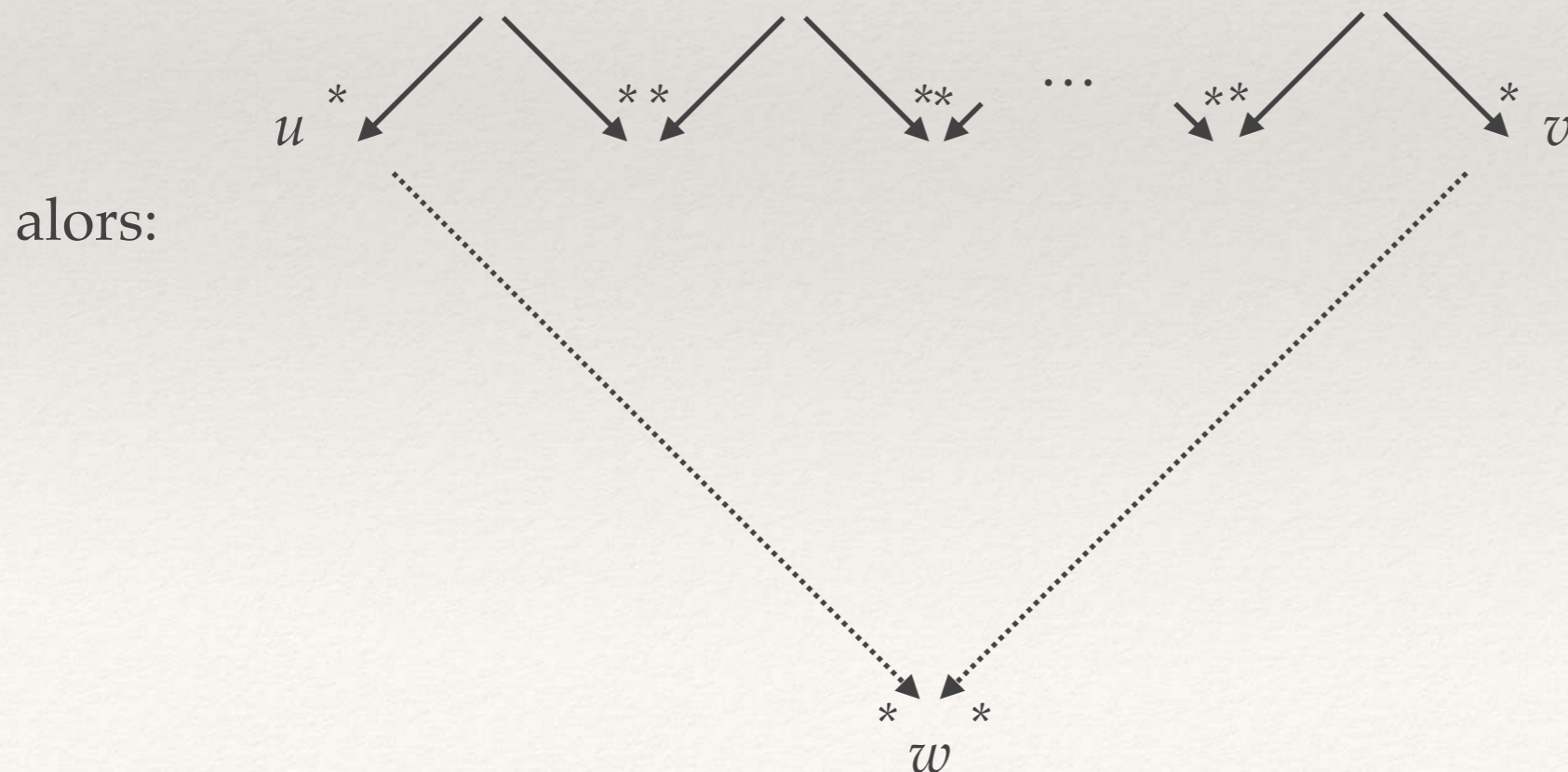
- ❖ Autrement dit, si:



Propriété de Church-Rosser

- ❖ \rightarrow a la propriété de **Church-Rosser** ssi $\leftrightarrow^* = \rightarrow^*; \leftarrow^*$
i.e., pour tous u, v tels que $u \leftrightarrow^* v$, il existe w tel que
$$u \rightarrow^* w \text{ et } w \leftarrow^* v$$

- ❖ Autrement dit, si:



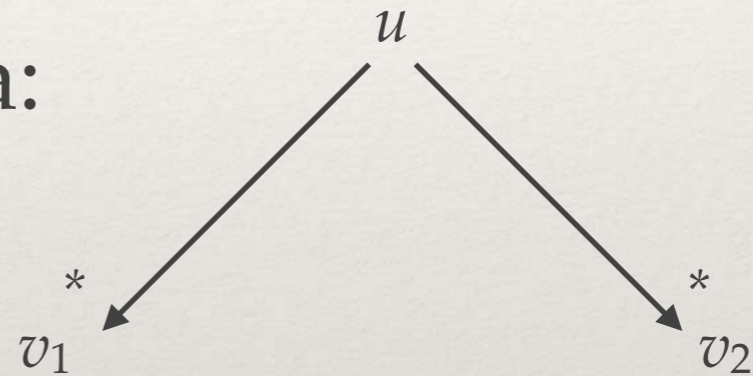
Confluence et Church-Rosser

- ❖ **Théorème.** Confluence = Church-Rosser.
- ❖ Preuve (1 / 2). Supposons \rightarrow Church-Rosser.

Si l'on a:

en particulier $v_1 \leftrightarrow^* v_2$

(par une preuve dite en **pic**)



Confluence et Church-Rosser

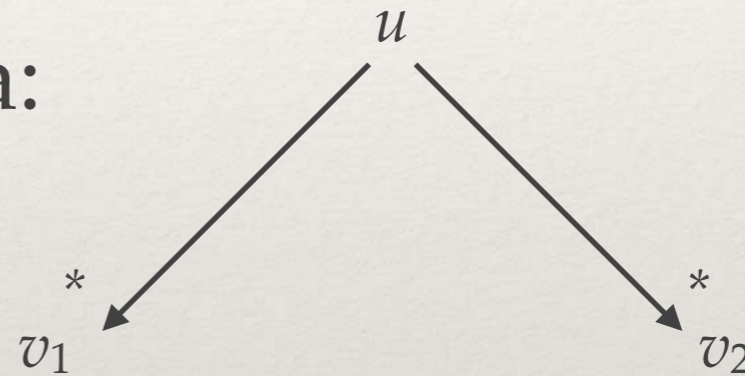
❖ **Théorème.** Confluence = Church-Rosser.

❖ Preuve (1 / 2). Supposons \rightarrow Church-Rosser.

Si l'on a:

en particulier $v_1 \leftrightarrow^* v_2$

(par une preuve dite en **pic**)



❖ Par Church-Rosser...

(preuve de $v_1 \leftrightarrow^* v_2$ dite en **vallée**)

Confluence et Church-Rosser

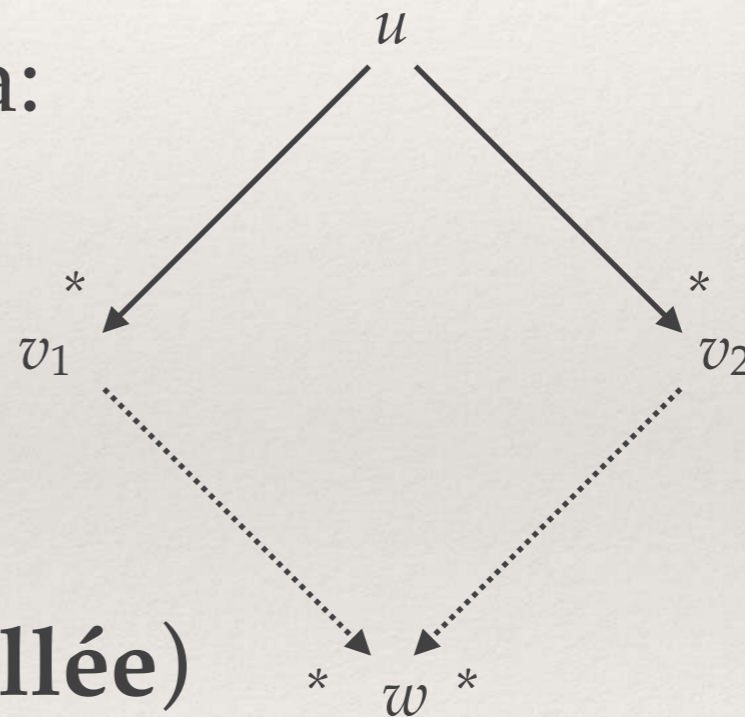
❖ **Théorème.** Confluence = Church-Rosser.

❖ Preuve (1 / 2). Supposons \rightarrow Church-Rosser.

Si l'on a:

en particulier $v_1 \leftrightarrow^* v_2$

(par une preuve dite en **pic**)



❖ Par Church-Rosser...

(preuve de $v_1 \leftrightarrow^* v_2$ dite en **vallée**)

❖ donc \rightarrow est confluente.

Confluence et Church-Rosser

❖ **Théorème.** Confluence = Church-Rosser.

❖ Preuve (2/2). Supposons \rightarrow confluente, et $u \leftrightarrow^* v$

On peut organiser cette preuve en pics et vallées:

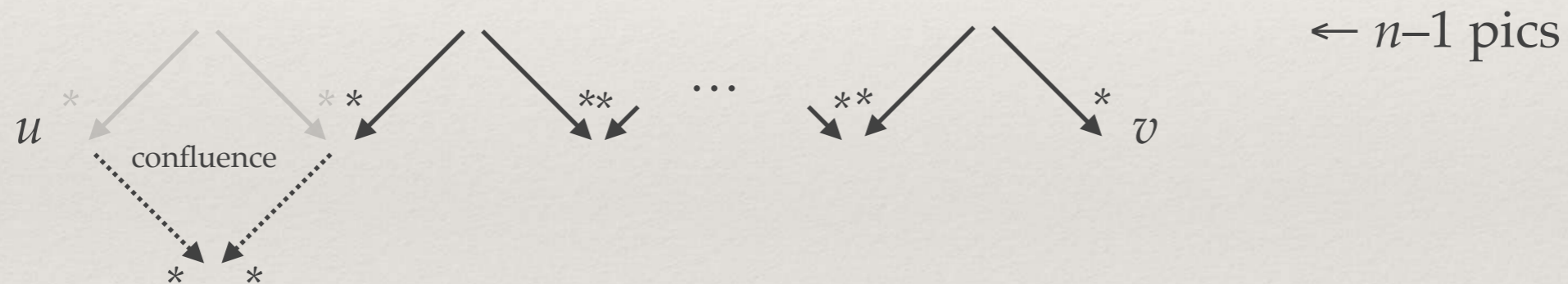


Confluence et Church-Rosser

❖ **Théorème.** Confluence = Church-Rosser.

❖ Preuve (2/2). Supposons \rightarrow confluente, et $u \leftrightarrow^* v$

On peut organiser cette preuve en pics et vallées:

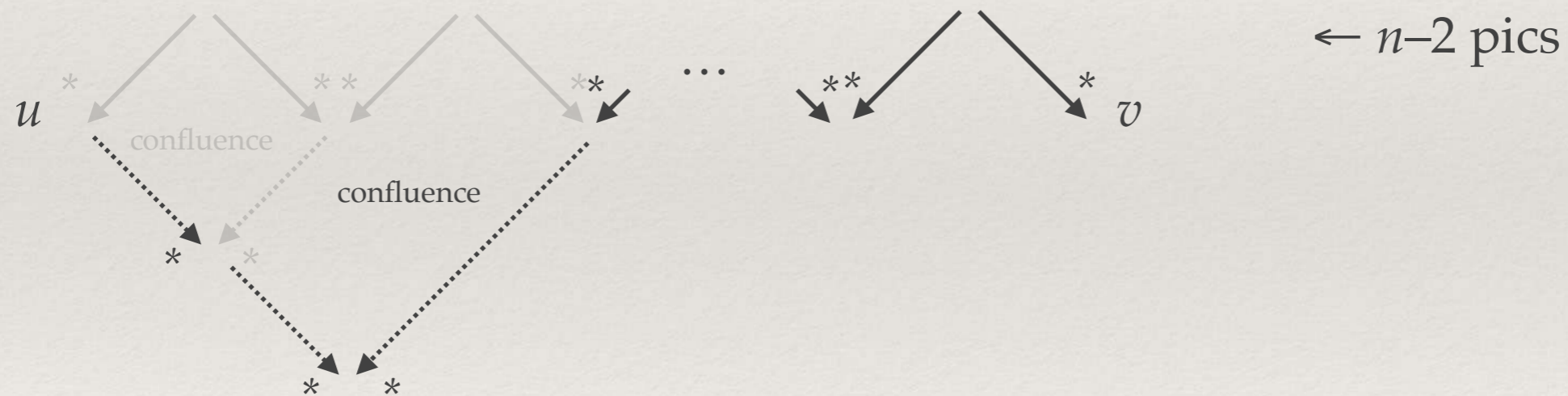


Confluence et Church-Rosser

❖ **Théorème.** Confluence = Church-Rosser.

❖ Preuve (2/2). Supposons \rightarrow confluente, et $u \leftrightarrow^* v$

On peut organiser cette preuve en pics et vallées:

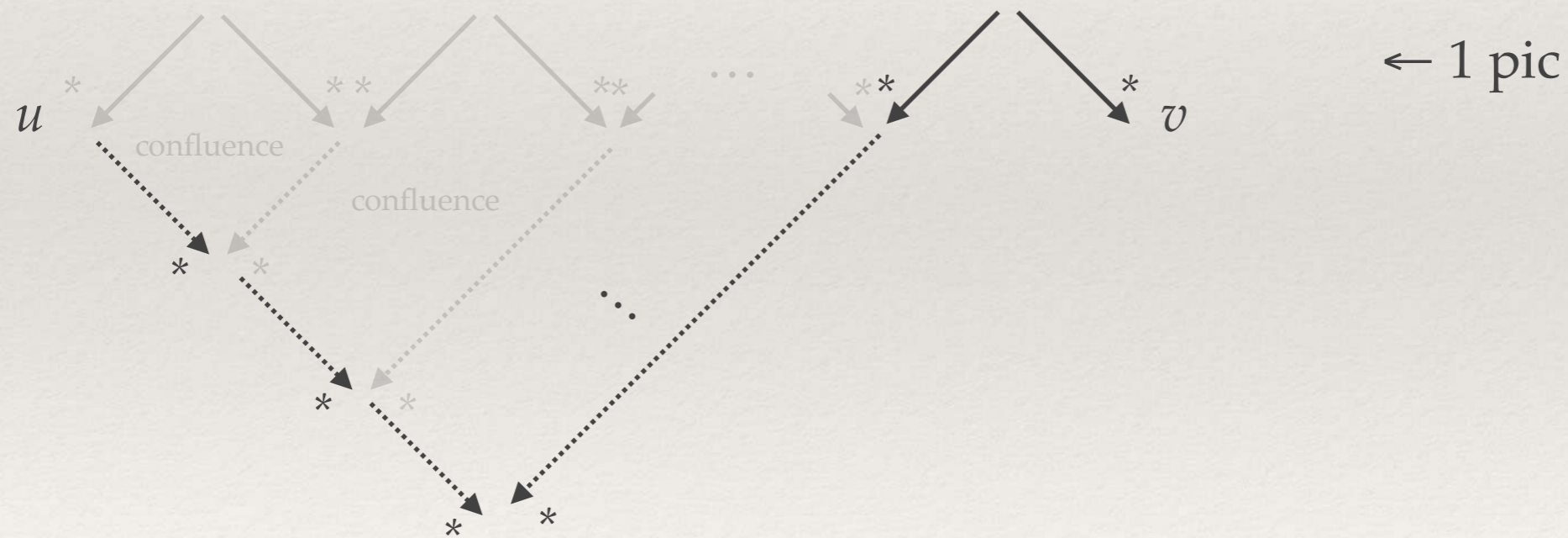


Confluence et Church-Rosser

❖ **Théorème.** Confluence = Church-Rosser.

❖ Preuve (2/2). Supposons \rightarrow confluente, et $u \leftrightarrow^* v$

On peut organiser cette preuve en pics et vallées:

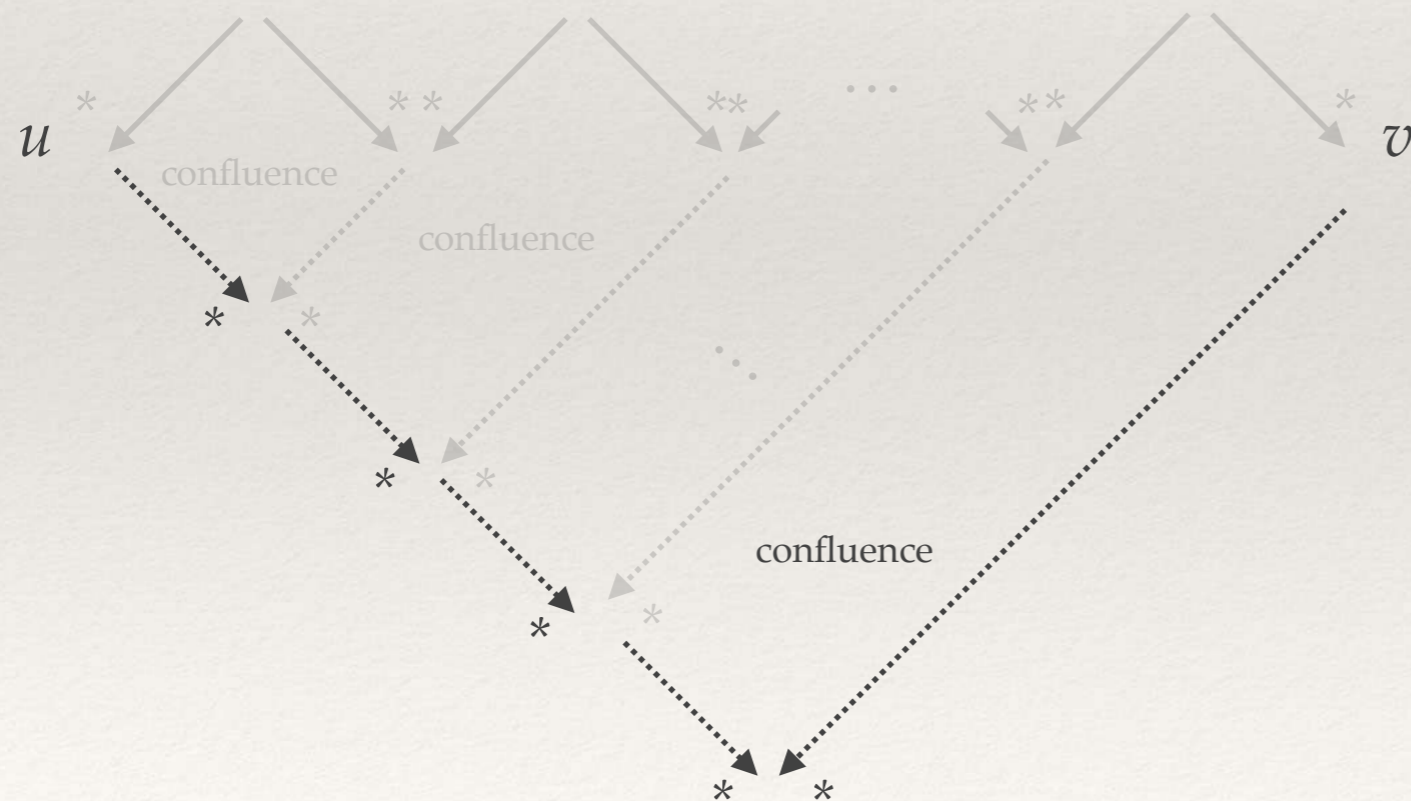


Confluence et Church-Rosser

❖ **Théorème.** Confluence = Church-Rosser.

❖ Preuve (2/2). Supposons \rightarrow confluente, et $u \leftrightarrow^* v$

On peut organiser cette preuve en pics et vallées:

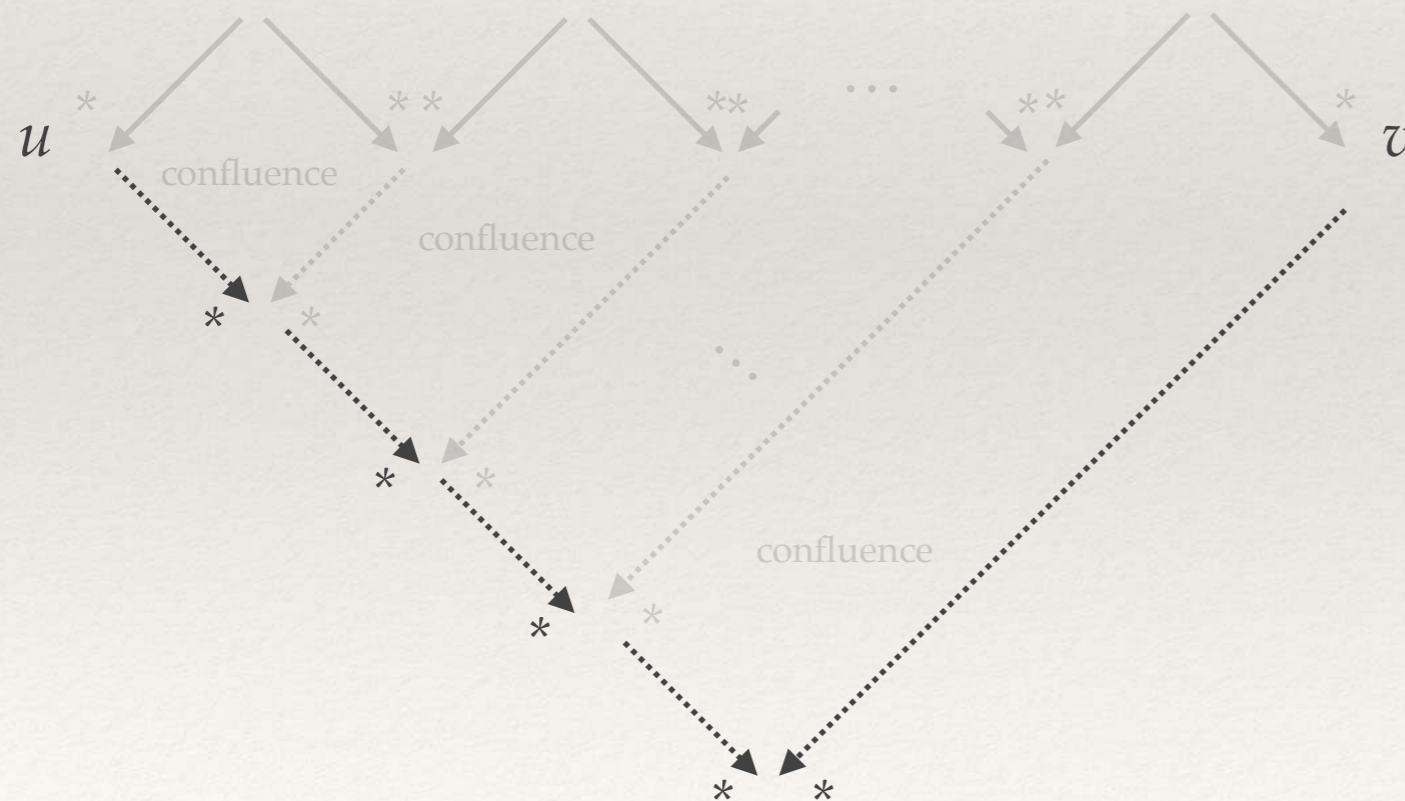


Confluence et Church-Rosser

❖ **Théorème.** Confluence = Church-Rosser.

❖ Preuve (2/2). Supposons \rightarrow confluente, et $u \leftrightarrow^* v$

On peut organiser cette preuve en pics et vallées:



Plus de pic!

On a obtenu une
preuve en vallée
de $u \leftrightarrow^* v$.

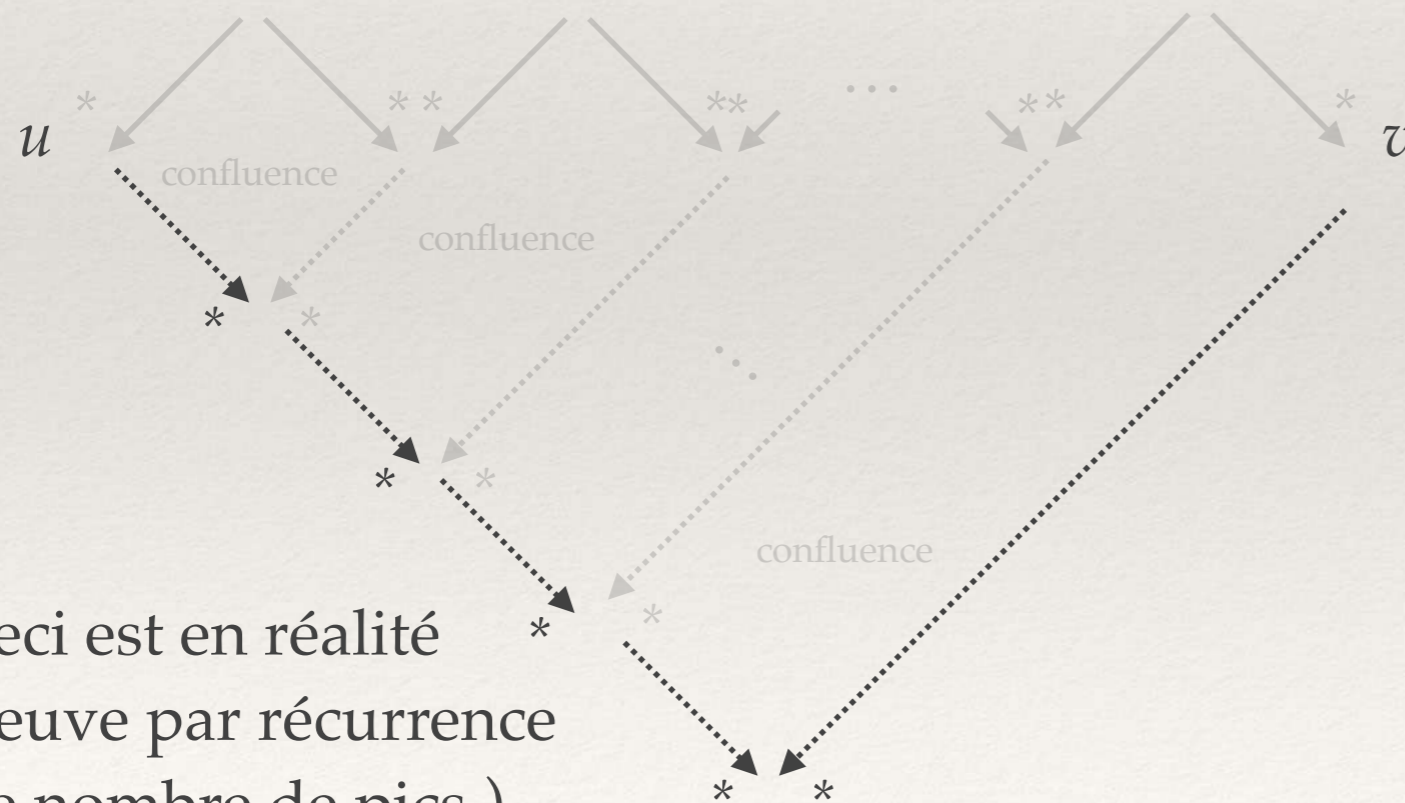
Donc \rightarrow est
Church-Rosser.

Confluence et Church-Rosser

❖ **Théorème.** Confluence = Church-Rosser.

❖ Preuve (2/2). Supposons \rightarrow confluente, et $u \leftrightarrow^* v$

On peut organiser cette preuve en pics et vallées:



(Ceci est en réalité
une preuve par récurrence
sur le nombre de pics.)

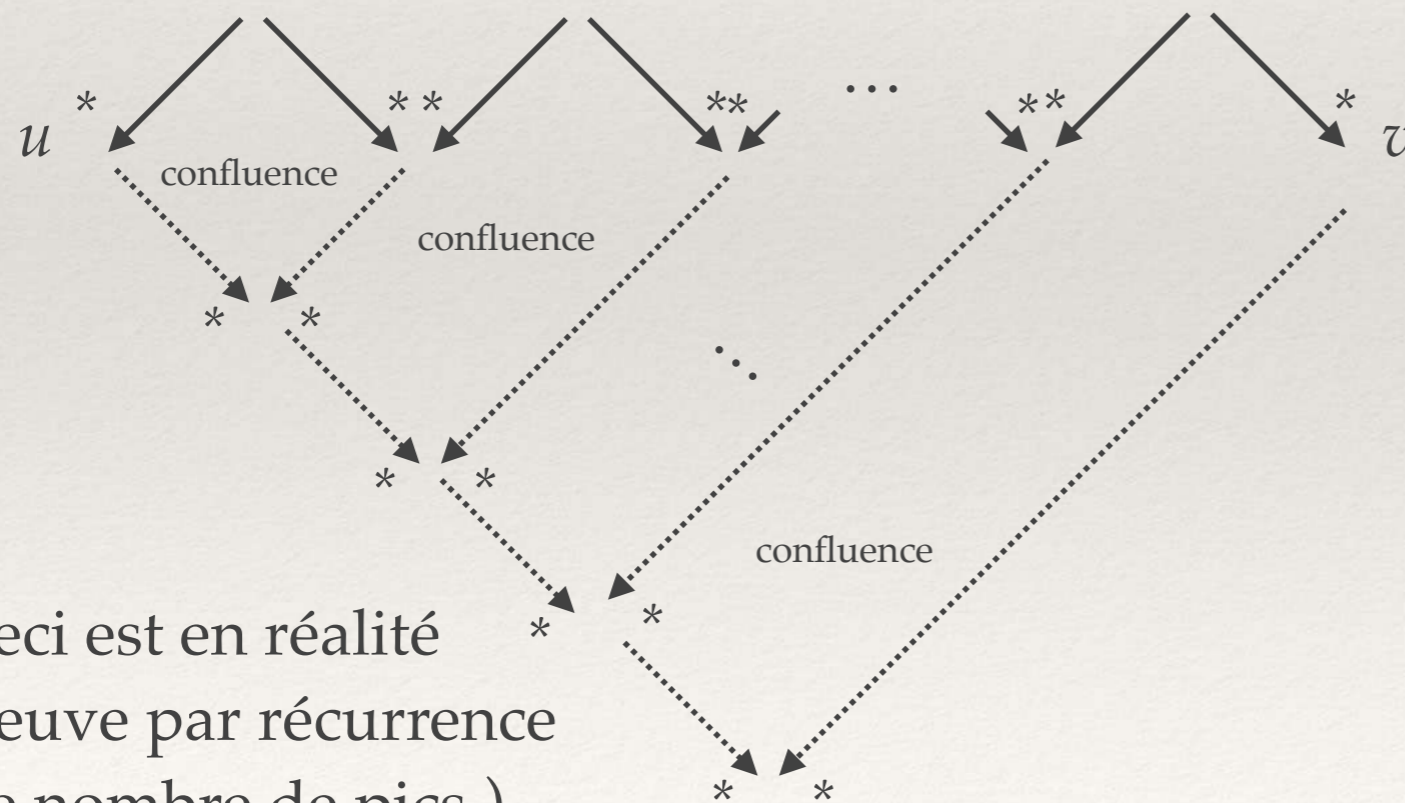
Plus de pic!
On a obtenu une
preuve en vallée
de $u \leftrightarrow^* v$.
Donc \rightarrow est
Church-Rosser.

Confluence et Church-Rosser

❖ **Théorème.** Confluence = Church-Rosser.

❖ Preuve (2/2). Supposons \rightarrow confluente, et $u \leftrightarrow^* v$

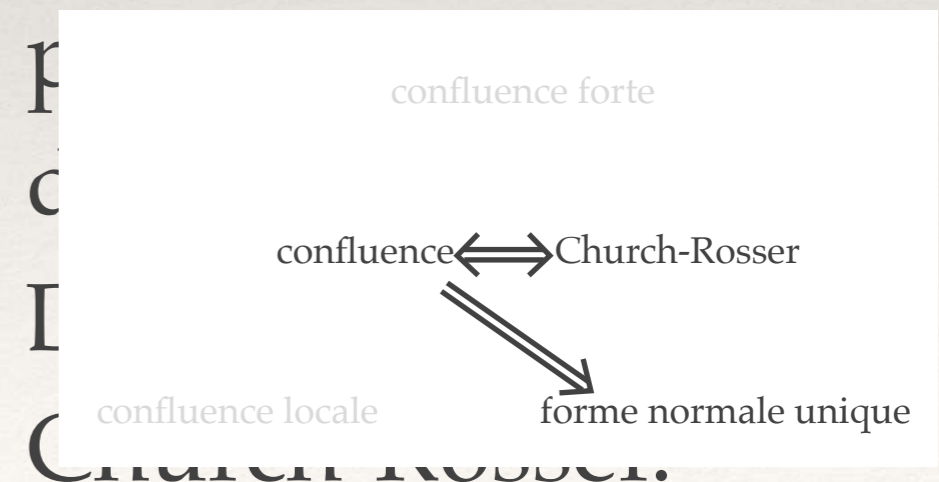
On peut organiser cette preuve en pics et vallées:



(Ceci est en réalité une preuve par récurrence sur le nombre de pics.)

$\leftarrow n$ pics
 $\leftarrow n-1$ pics
 $\leftarrow 1$ pic

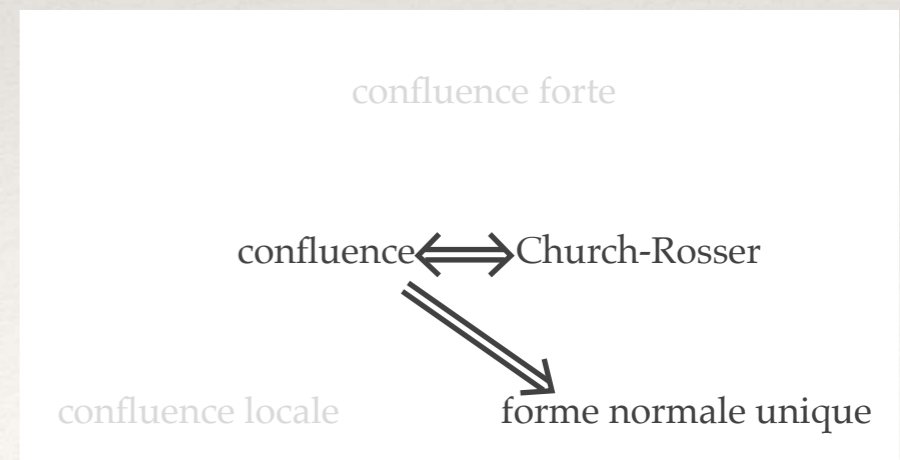
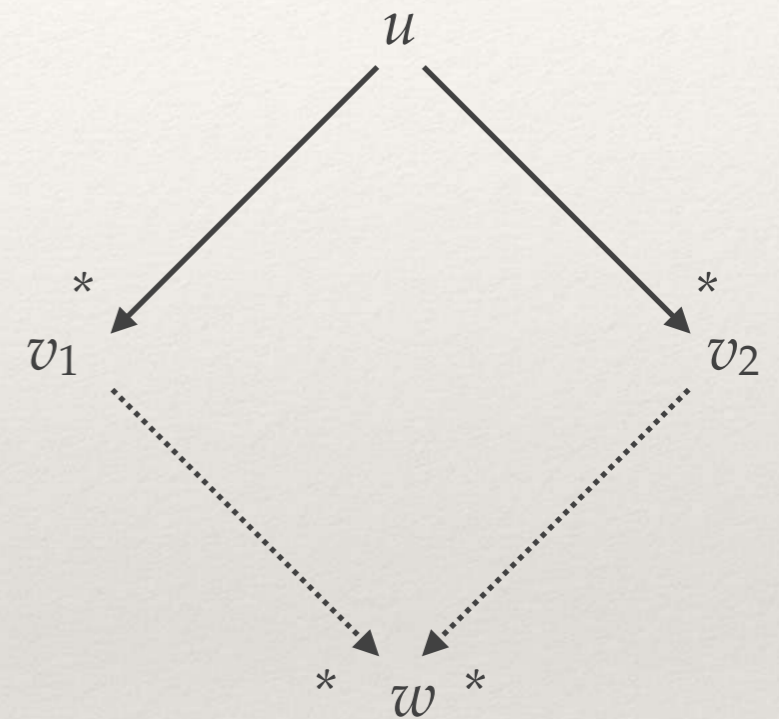
Plus de pic!
 On a obtenu une



CHURCH-ROSSER.

Comment prouver la confluence?

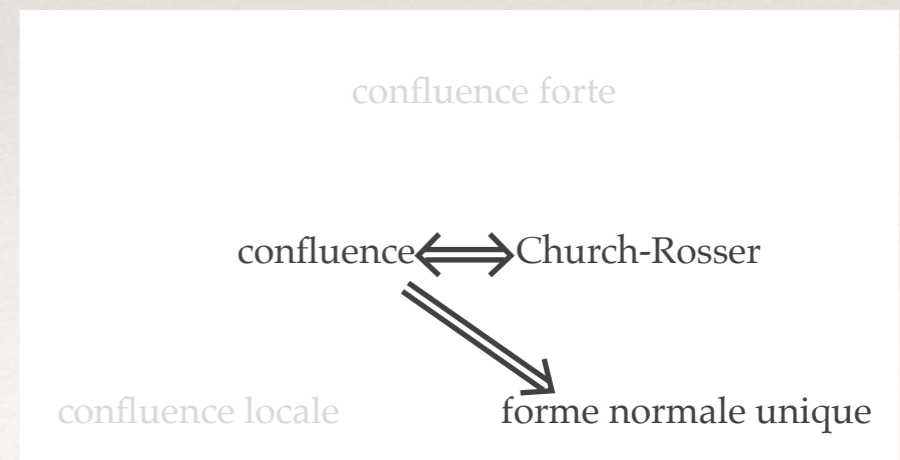
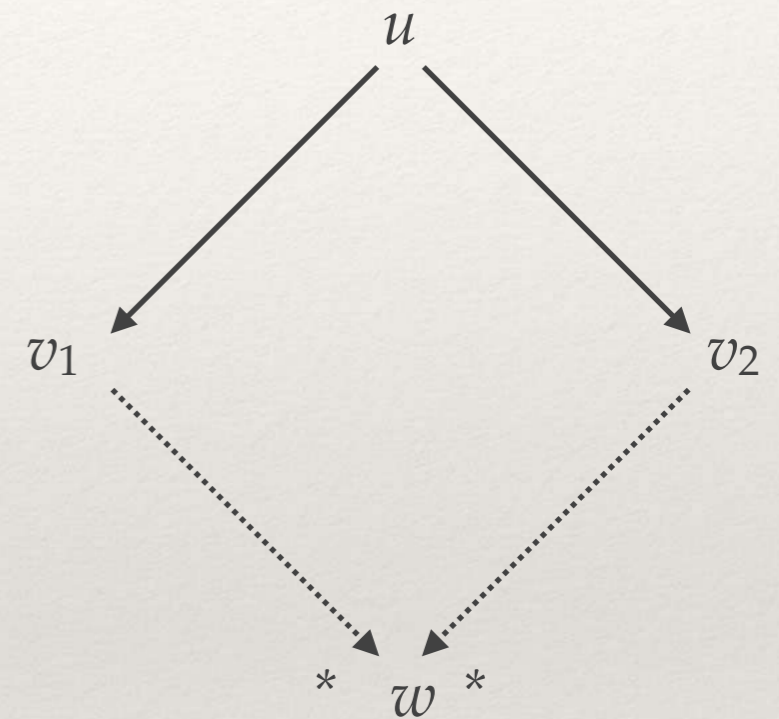
- ❖ En principe, on devrait énumérer toutes les situations $v_1 \leftarrow^* u \rightarrow^* v_2 \dots$
- ❖ ... pour toutes les longueurs de réduction possibles de u à v_1 (resp. v_2)
- ❖ On va donc chercher des critères de confluence plus simples



Confluence forte, confluence locale

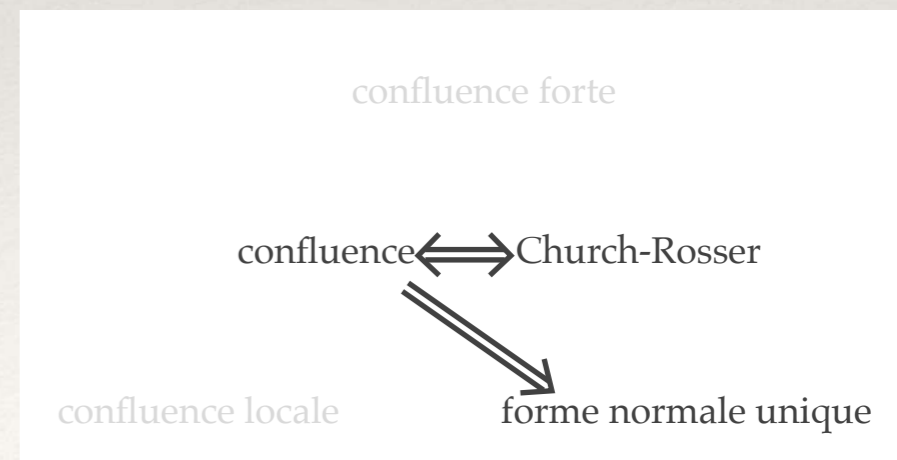
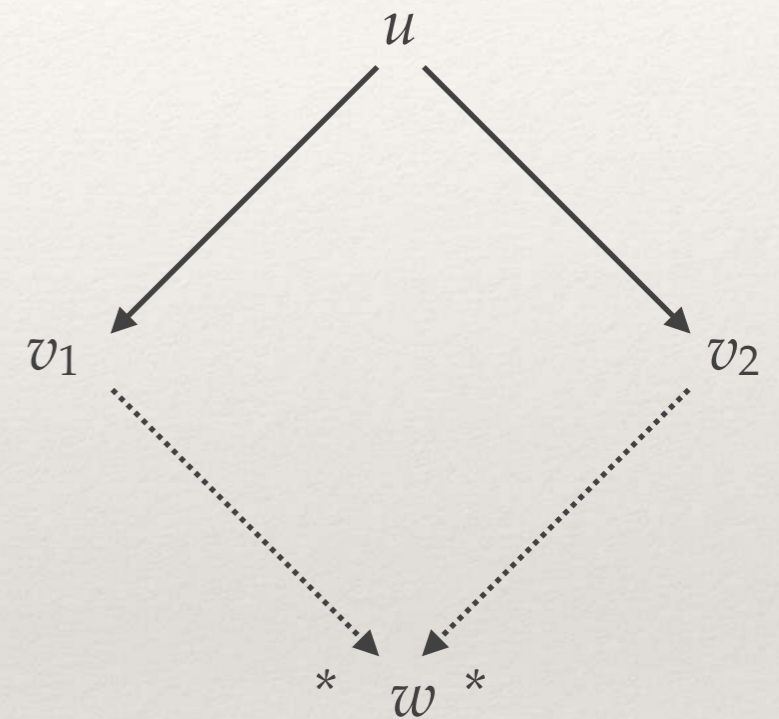
Confluence locale

- ❖ \rightarrow est localement confluente ssi:
- ❖ Voyez-vous la différence avec la confluence?



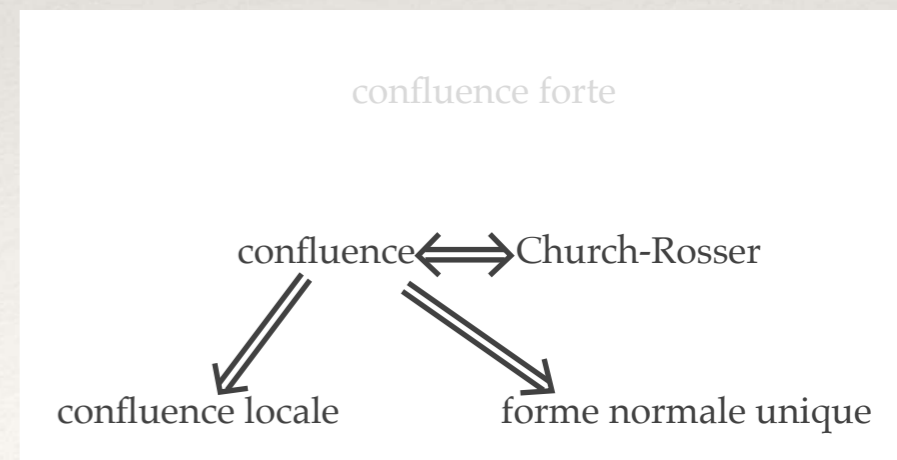
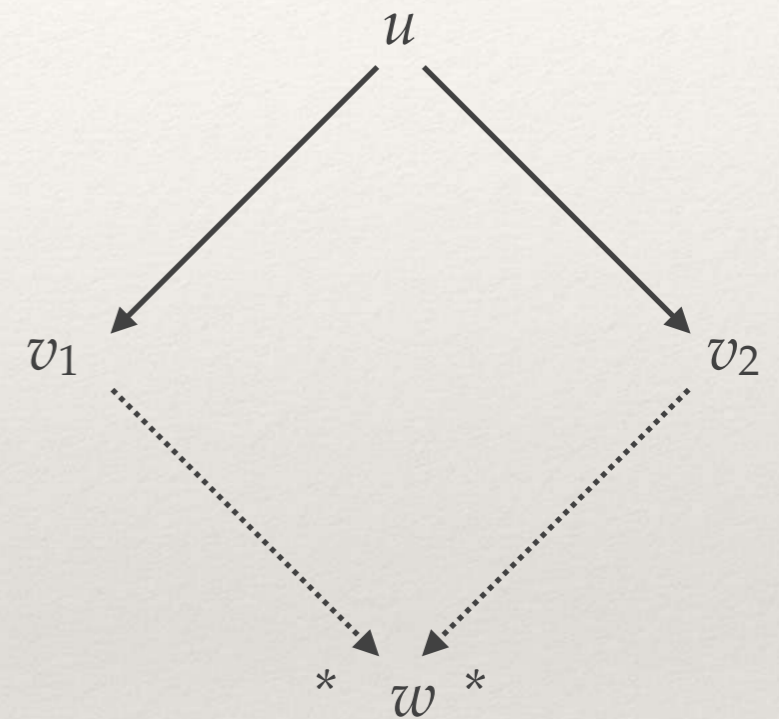
Confluence locale

- ❖ \rightarrow est **localement confluente** ssi:
- ❖ Voyez-vous la différence avec la confluence?
- ❖ Plus facile à vérifier (en fait, on n'a qu'à énumérer les paires critiques)



Confluence locale

- ❖ \rightarrow est localement confluente ssi:
- ❖ Voyez-vous la différence avec la confluence?
- ❖ Plus facile à vérifier (en fait, on n'a qu'à énumérer les paires critiques)
- ❖ **Fait.** Confluence implique confluence locale.

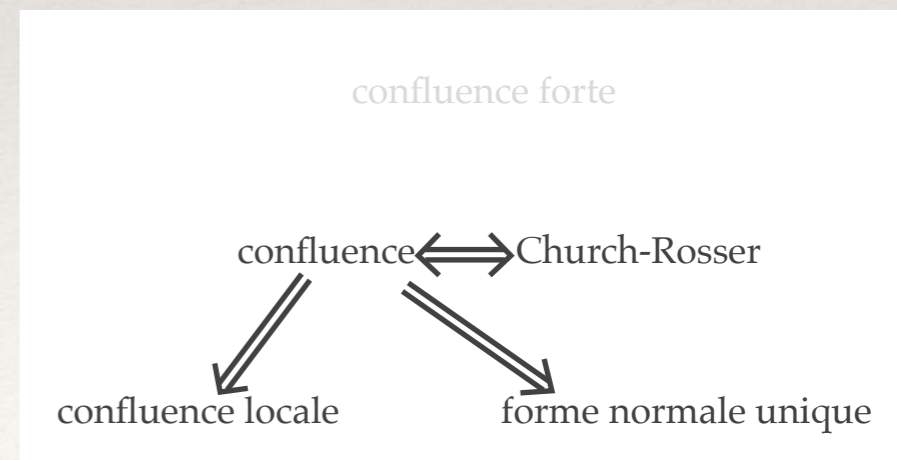
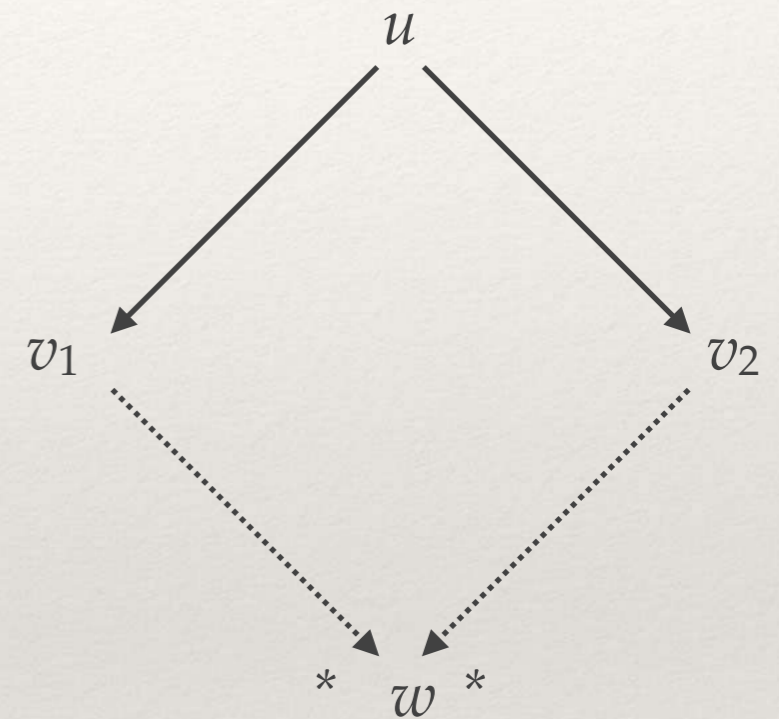


Confluence locale

- ❖ \rightarrow est localement confluente ssi:
- ❖ Voyez-vous la différence avec la confluence?
- ❖ Plus facile à vérifier (en fait, on n'a qu'à énumérer les paires critiques)

❖ **Fait.** Confluence implique confluence locale.

(L'implication n'est pas dans le sens souhaité... ça arrive.)

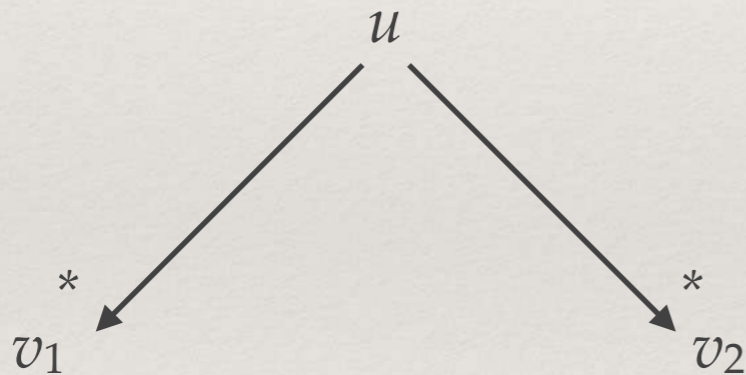


Confluence locale et confluence?

- ❖ La preuve suivante est fausse, dites-moi pourquoi.
- ❖ **Arnaque.** Si \rightarrow loc. confluente alors \rightarrow confluente (non).

Confluence locale et confluence?

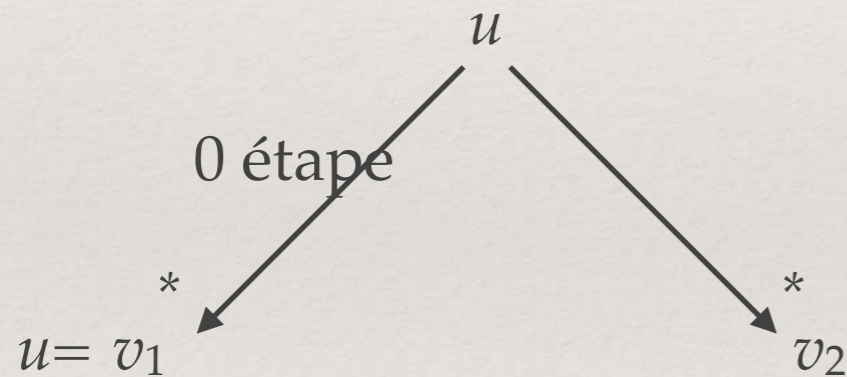
- ❖ La preuve suivante est fausse, dites-moi pourquoi.
- ❖ **Arnaque.** Si \rightarrow loc. confluente alors \rightarrow confluente (non).
- ❖ Supposons:



Confluence locale et confluence?

- ❖ La preuve suivante est fausse, dites-moi pourquoi.
- ❖ **Arnaque.** Si \rightarrow loc. confluente alors \rightarrow confluente (non).
- ❖ Supposons:

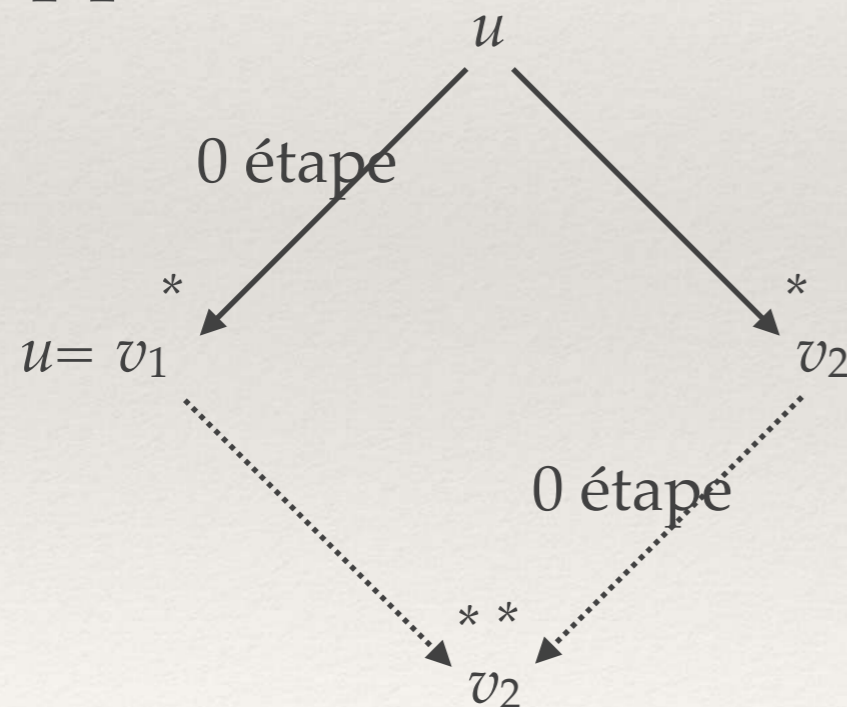
Si $u = v_1 \dots$



Confluence locale et confluence?

- ❖ La preuve suivante est fausse, dites-moi pourquoi.
- ❖ **Arnaque.** Si \rightarrow loc. confluente alors \rightarrow confluente (non).
- ❖ Supposons:

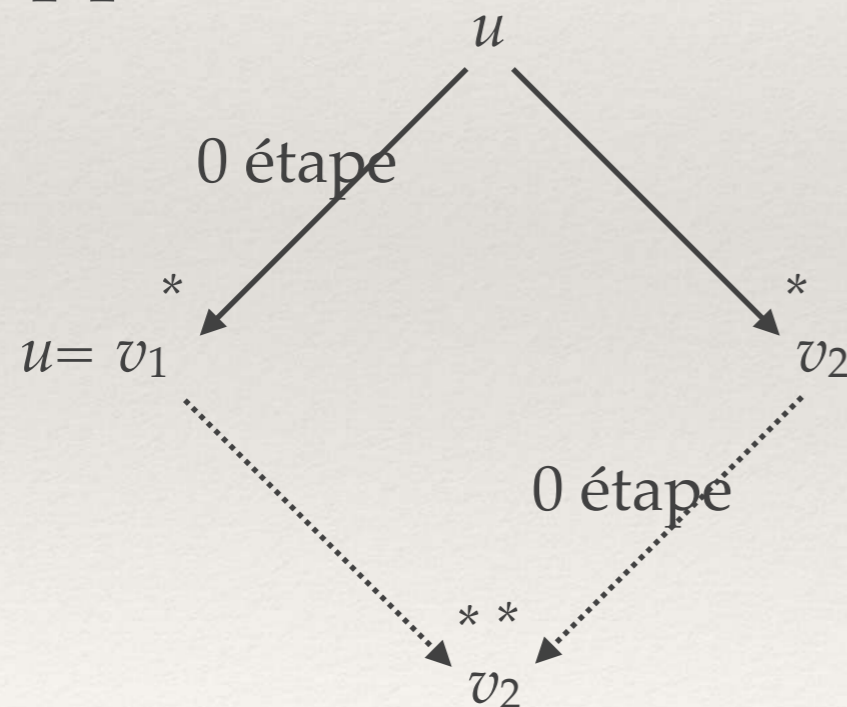
Si $u=v_1\dots$



Confluence locale et confluence?

- ❖ La preuve suivante est fausse, dites-moi pourquoi.
- ❖ **Arnaque.** Si \rightarrow loc. confluente alors \rightarrow confluente (non).

- ❖ Supposons:



Si $u=v_1 \dots$

Si $u=v_2$, on raisonne
symétriquement.

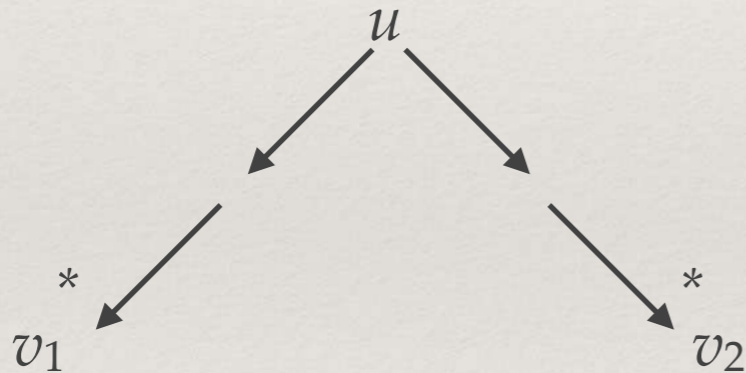
Regardons donc ce qui se
passe si ≥ 1 étape de u à v_1 ,
resp. à v_2

Confluence locale et confluence?

- ❖ La preuve suivante est fausse, dites-moi pourquoi.
- ❖ **Arnaque.** Si \rightarrow loc. confluente alors \rightarrow confluente (non).

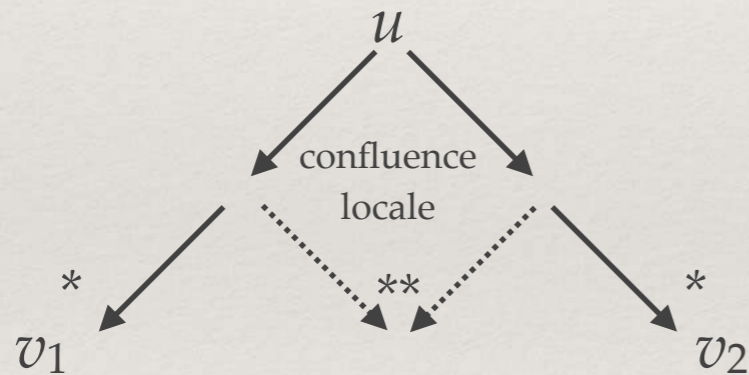
Confluence locale et confluence?

- ❖ La preuve suivante est fausse, dites-moi pourquoi.
- ❖ **Arnaque.** Si \rightarrow loc. confluente alors \rightarrow confluente (non).
- ❖ Cas de récurrence:



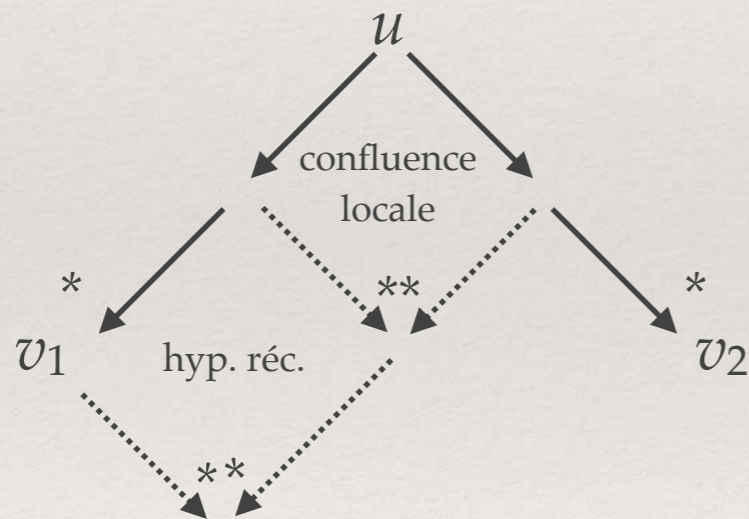
Confluence locale et confluence?

- ❖ La preuve suivante est fausse, dites-moi pourquoi.
- ❖ **Arnaque.** Si \rightarrow loc. confluente alors \rightarrow confluente (non).
- ❖ Cas de récurrence:



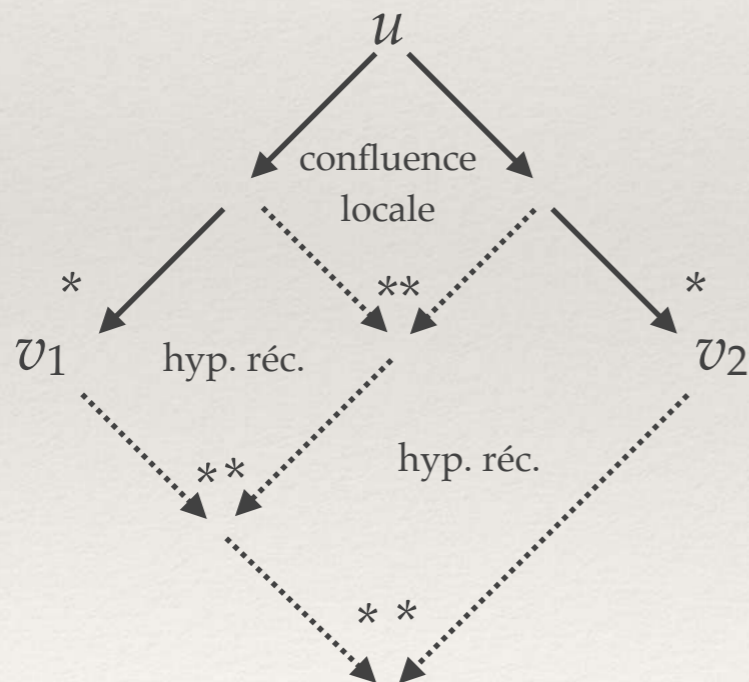
Confluence locale et confluence?

- ❖ La preuve suivante est fautive, dites-moi pourquoi.
- ❖ **Arnaque.** Si \rightarrow loc. confluente alors \rightarrow confluente (non).
- ❖ Cas de récurrence:



Confluence locale et confluence?

- ❖ La preuve suivante est fautive, dites-moi pourquoi.
- ❖ **Arnaque.** Si \rightarrow loc. confluente alors \rightarrow confluente (non).
- ❖ Cas de récurrence:



Où est l'erreur?

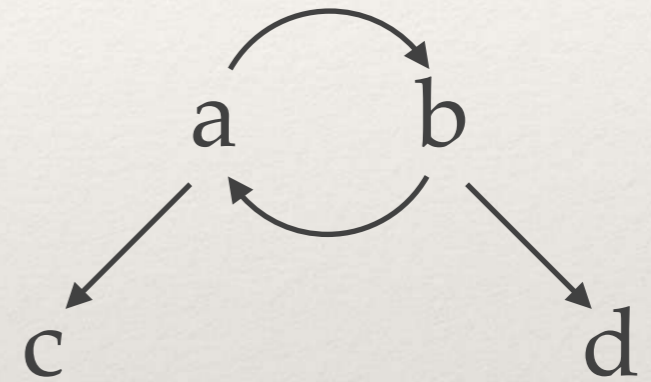
Le contre-exemple de Curry



Haskell B. Curry

<https://upload.wikimedia.org/wikipedia/commons/8/86/HaskellBCurry.jpg>

- ❖ Une relation \rightarrow définie sur un ensemble à 4 éléments $\{a,b,c,d\}$



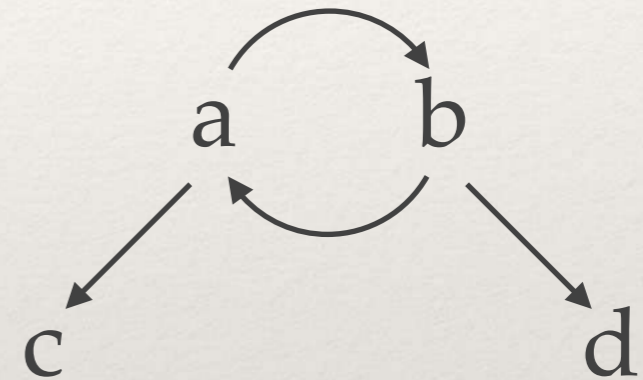
Le contre-exemple de Curry



Haskell B. Curry

<https://upload.wikimedia.org/wikipedia/commons/8/86/HaskellBCurry.jpg>

- ❖ Une relation \rightarrow définie sur un ensemble à 4 éléments $\{a,b,c,d\}$
- ❖ Localement confluent:
 - $c \leftarrow a \rightarrow b$ joignable
 - $a \leftarrow b \rightarrow d$ joignable



Le contre-exemple de Curry



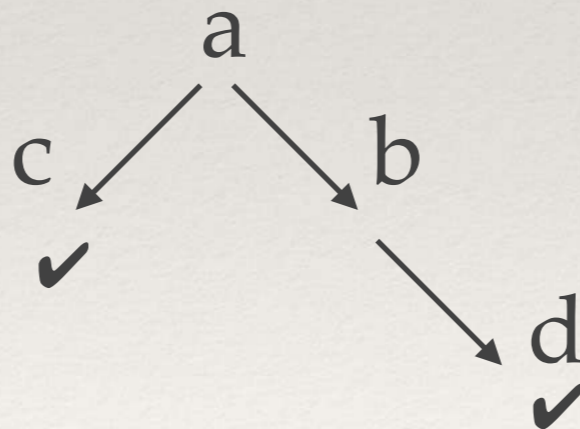
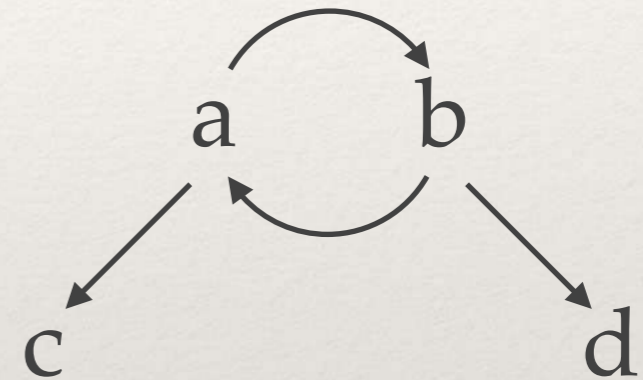
Haskell B. Curry

<https://upload.wikimedia.org/wikipedia/commons/8/86/HaskellBCurry.jpg>

❖ Une relation \rightarrow définie sur un ensemble à 4 éléments $\{a,b,c,d\}$

❖ Localement confluent:
— $c \leftarrow a \rightarrow b$ joignable
— $a \leftarrow b \rightarrow d$ joignable

❖ Pas confluent
(a n'a pas de forme normale unique)



Le contre-exemple de Curry



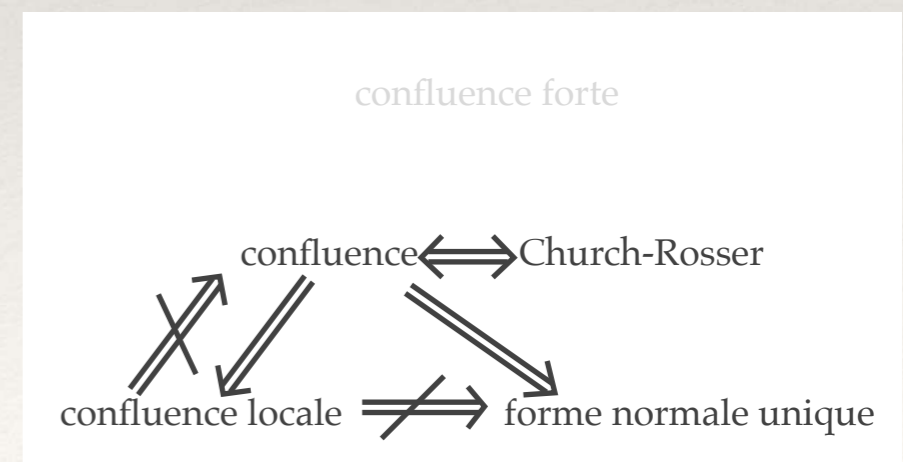
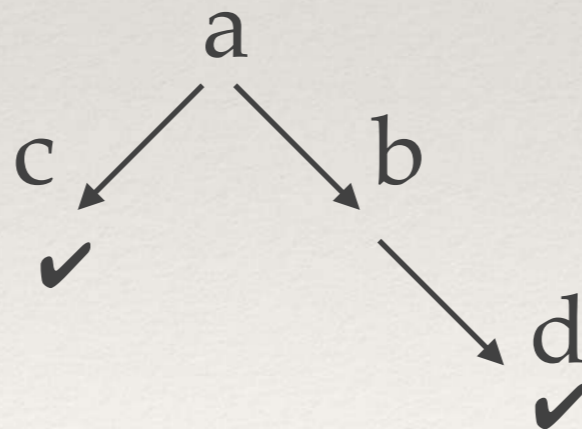
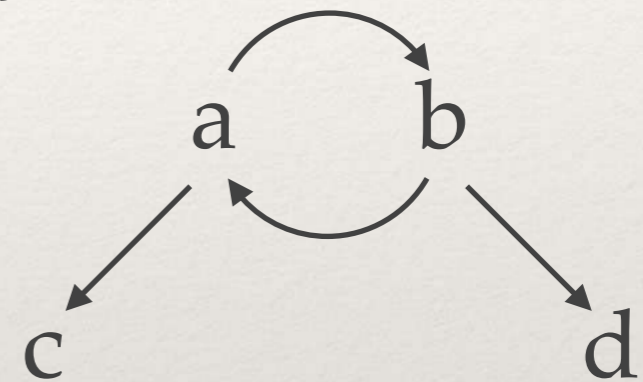
Haskell B. Curry

<https://upload.wikimedia.org/wikipedia/commons/8/86/HaskellBCurry.jpg>

❖ Une relation \rightarrow définie sur un ensemble à 4 éléments $\{a,b,c,d\}$

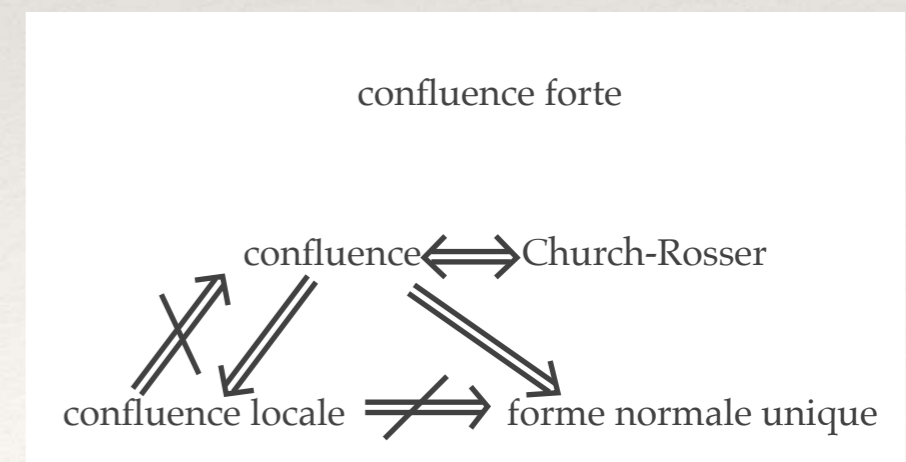
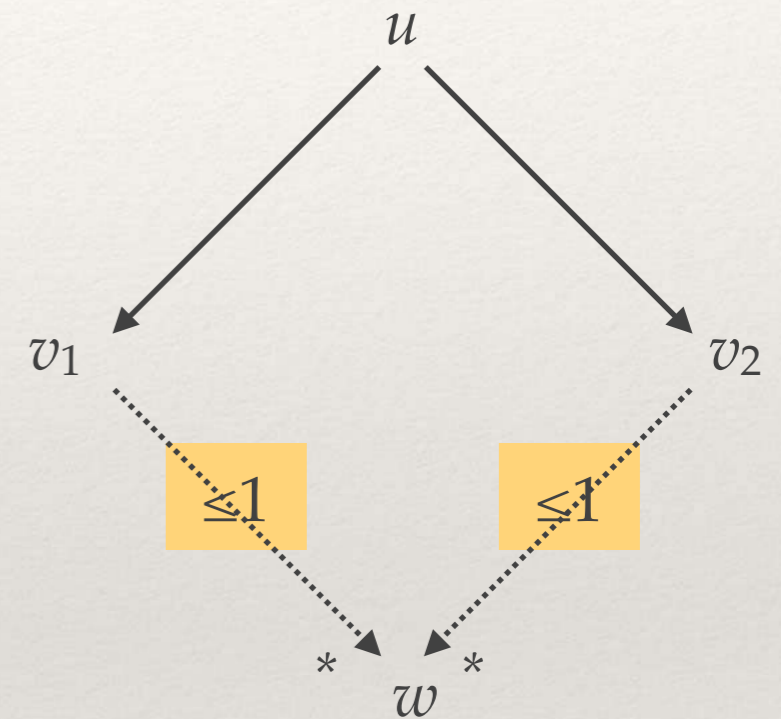
❖ Localement confluent:
— $c \leftarrow a \rightarrow b$ joignable
— $a \leftarrow b \rightarrow d$ joignable

❖ Pas confluent
(a n'a pas de forme normale unique)



Confluence forte

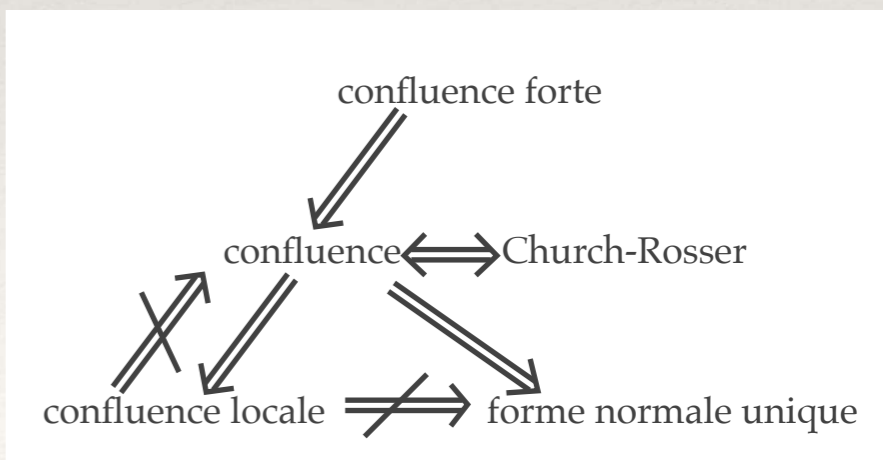
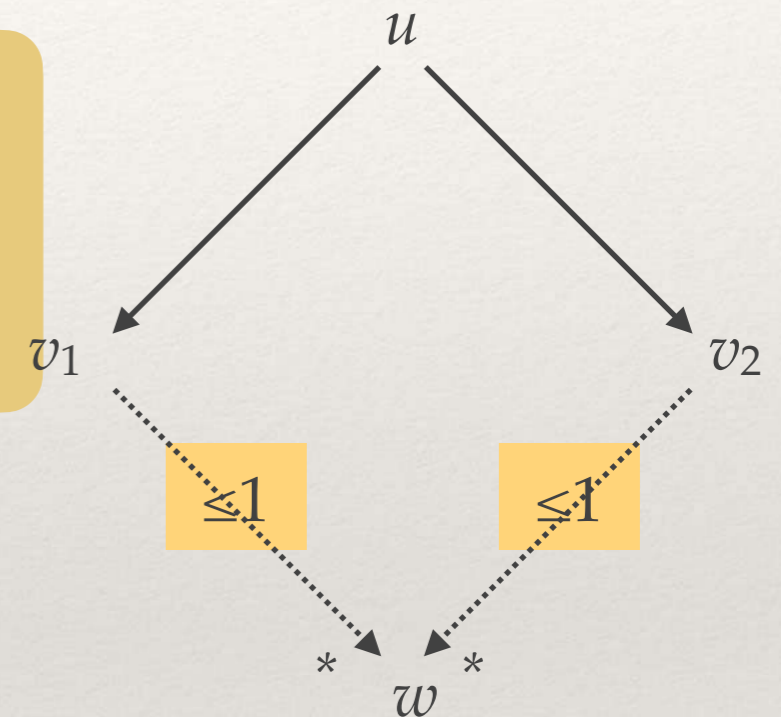
❖ \rightarrow est fortement confluente ssi:



Confluence forte

❖ \rightarrow est fortement confluente ssi:

❖ **Lemme.** La confluence forte implique la confluence.

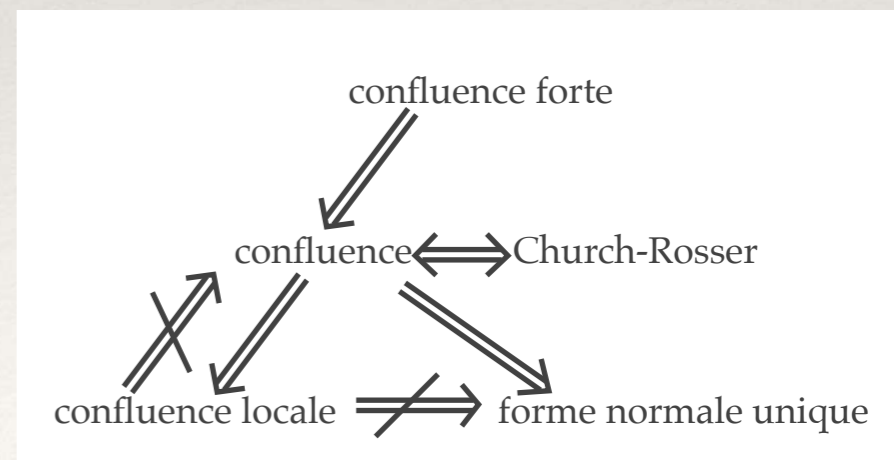
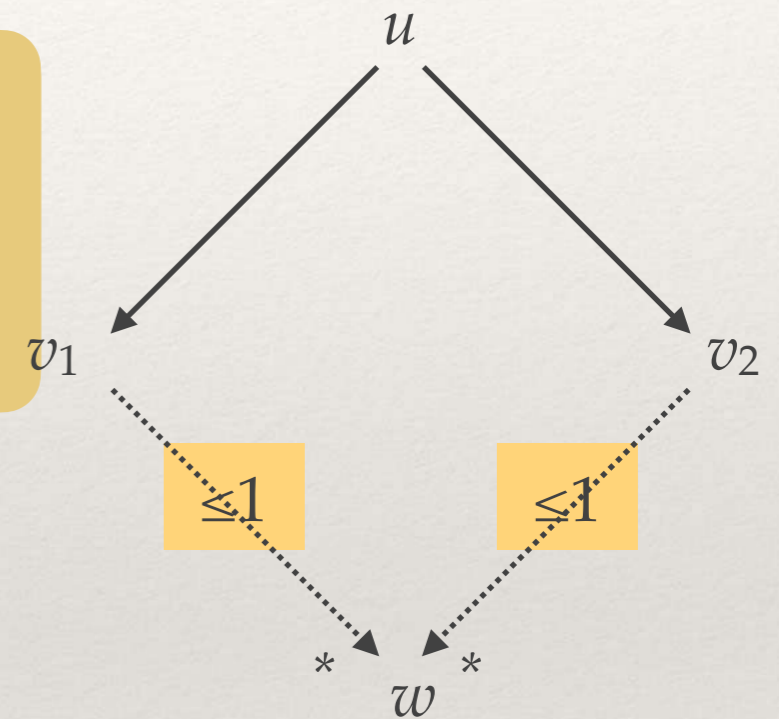


Confluence forte

❖ \rightarrow est fortement confluente ssi:

❖ **Lemme.** La confluence forte implique la confluence.

❖ Preuve: *quasiment* comme avant



Confluence forte

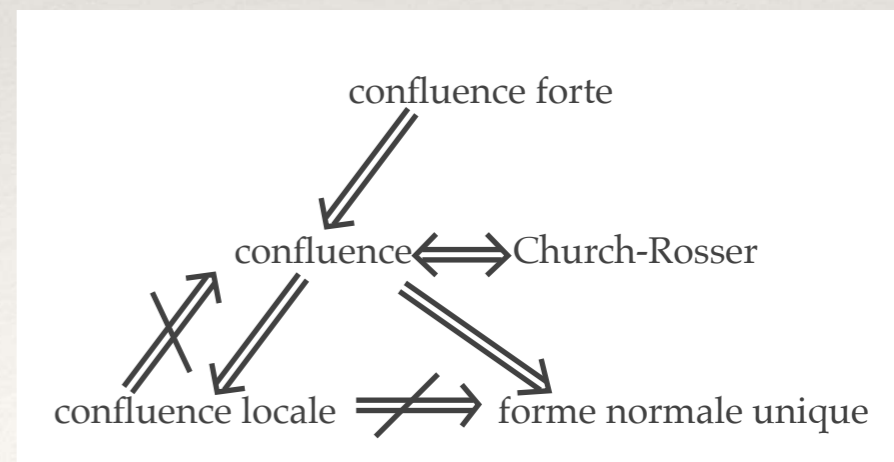
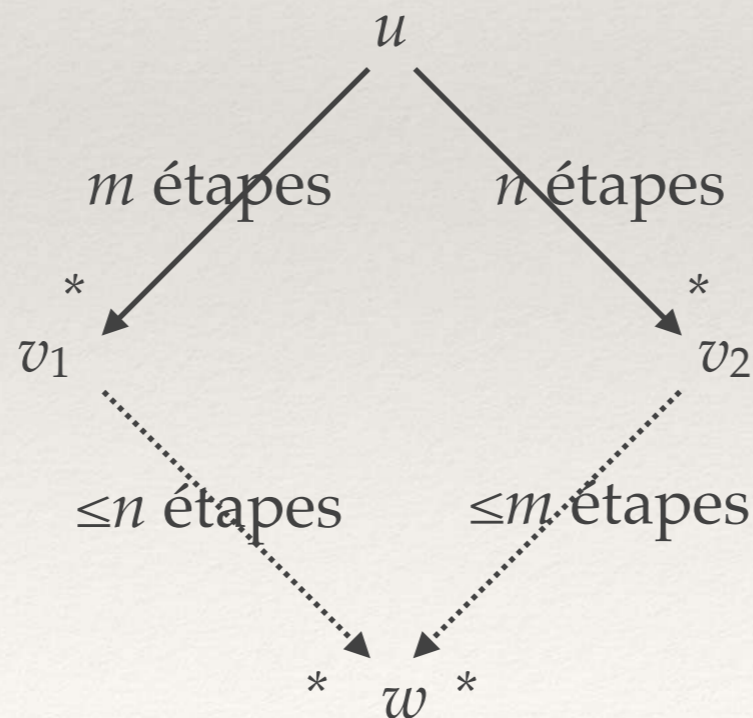
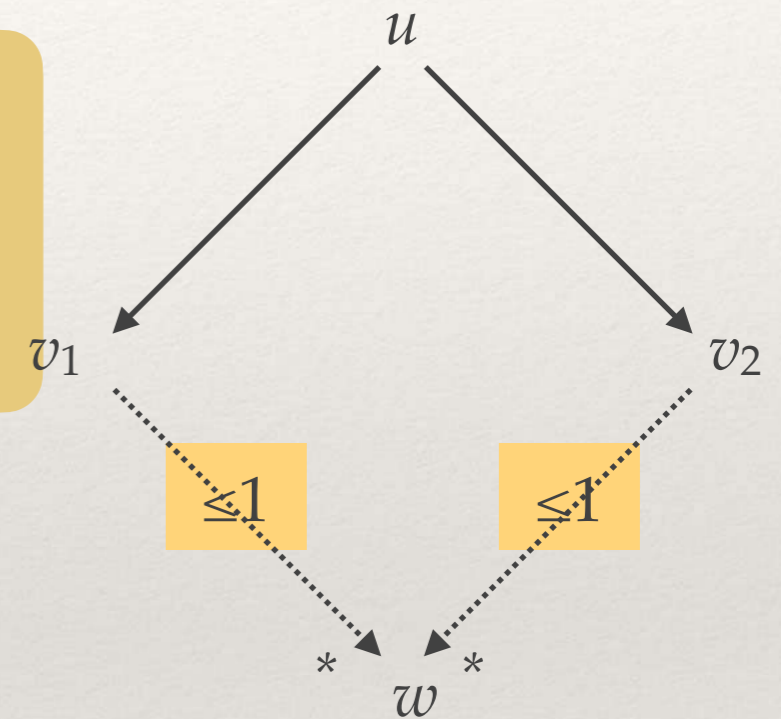
❖ \rightarrow est **fortement confluente** ssi:

❖ **Lemme.** La confluence forte implique la confluence.

❖ Preuve: *quasiment* comme avant

❖ On montre:

❖ par récurrence sur $m+n$.



Confluence forte

❖ \rightarrow est **fortement confluente** ssi:

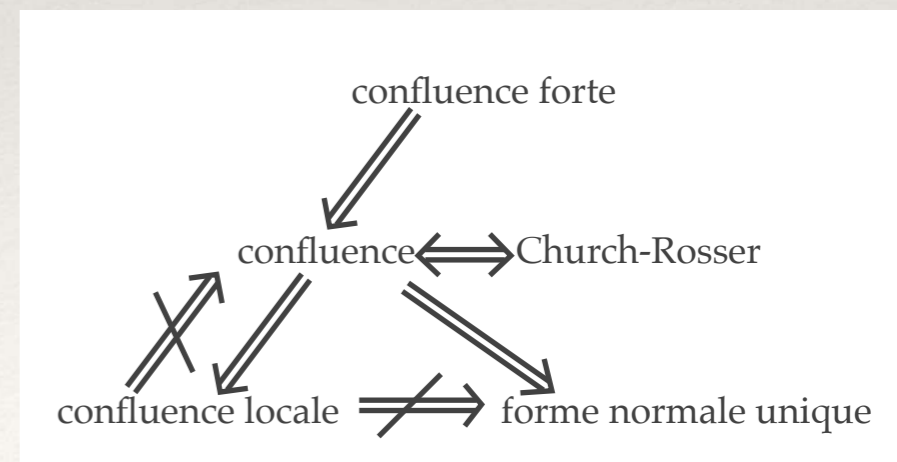
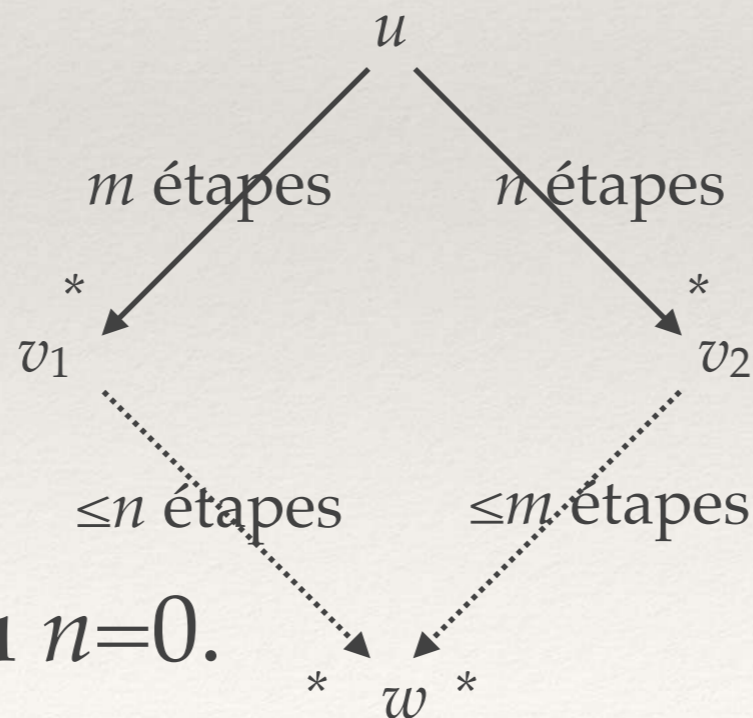
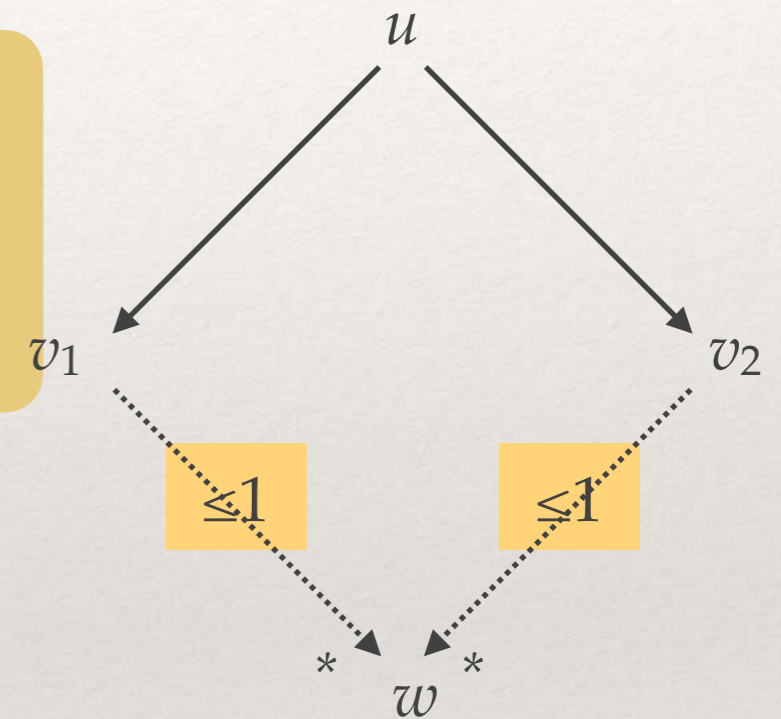
❖ **Lemme.** La confluence forte implique la confluence.

❖ Preuve: *quasiment* comme avant

❖ On montre:

❖ par récurrence sur $m+n$.

❖ Évident si $m=0$ ou $n=0$.

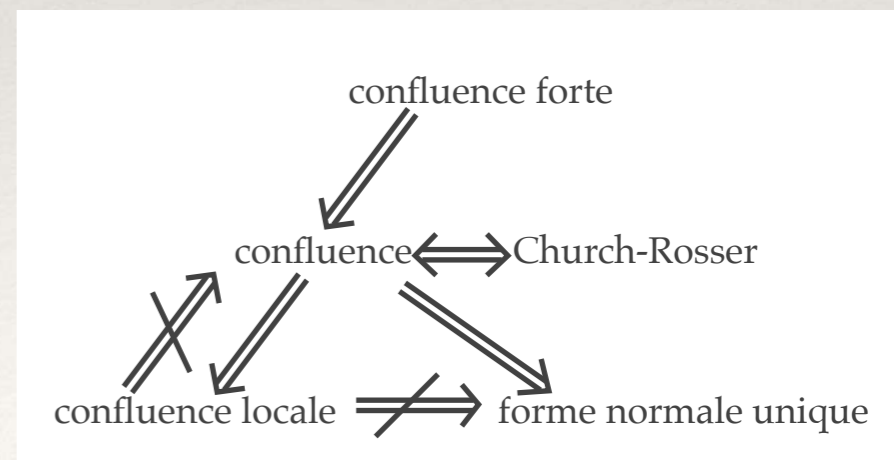
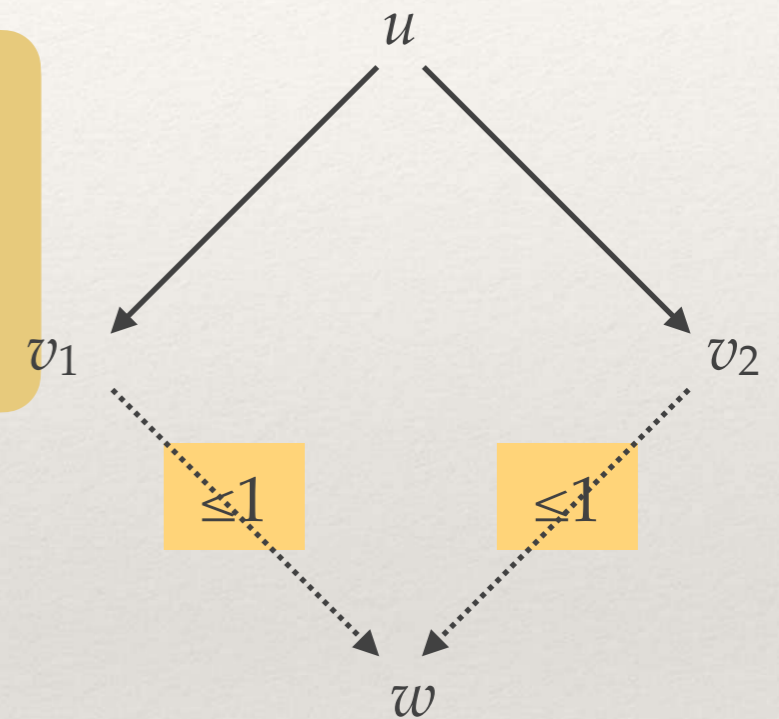
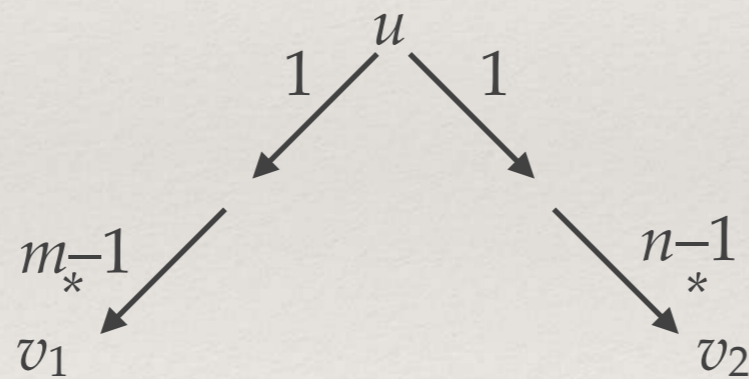


Confluence forte

❖ \rightarrow est fortement confluente ssi:

❖ **Lemme.** La confluence forte implique la confluence.

❖ Si $m \geq 1$ et $n \geq 1$:

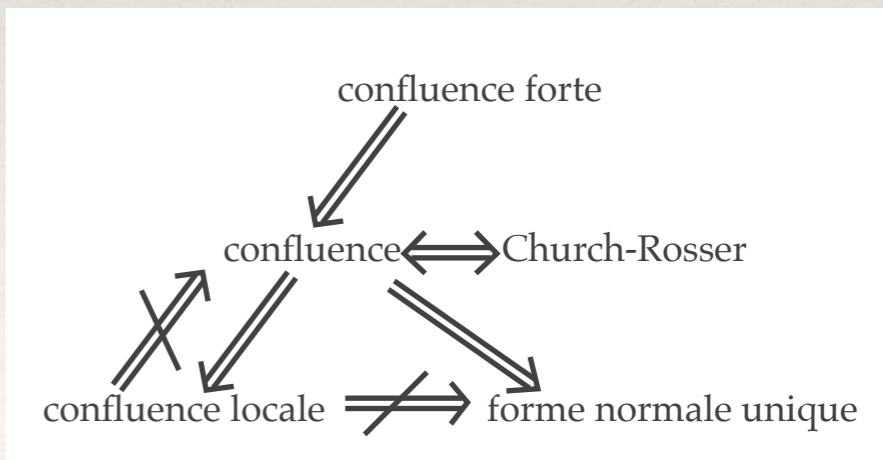
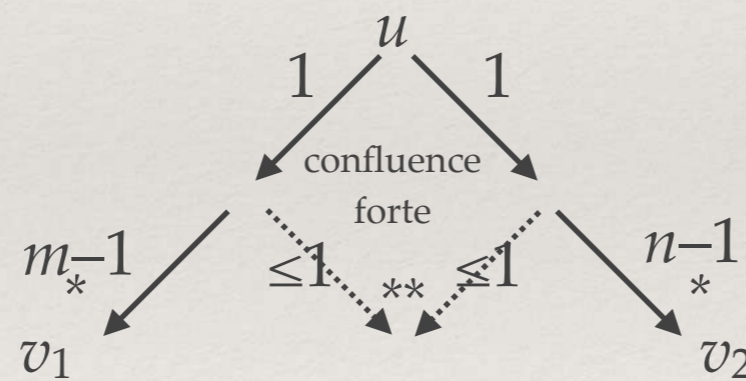
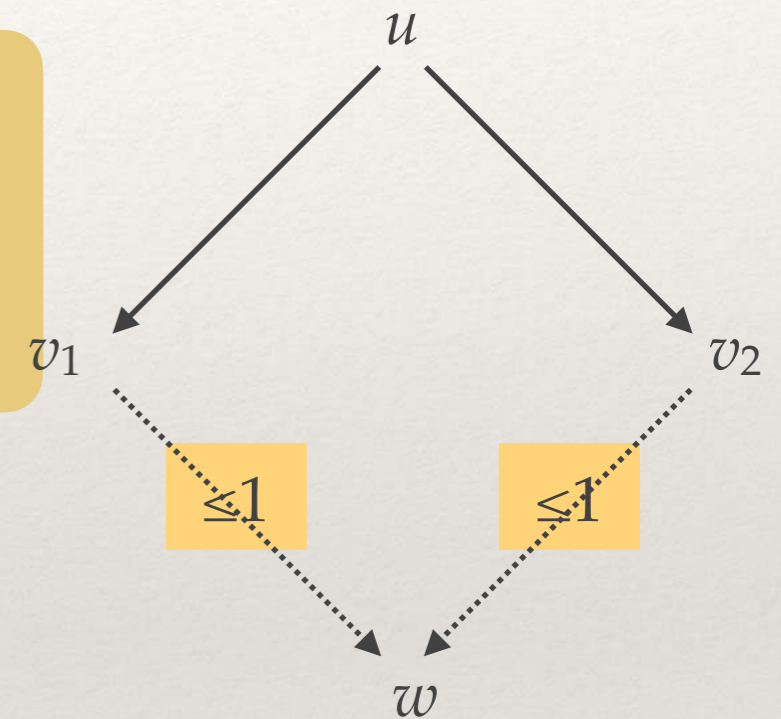


Confluence forte

❖ \rightarrow est fortement confluente ssi:

❖ **Lemme.** La confluence forte implique la confluence.

❖ Si $m \geq 1$ et $n \geq 1$:



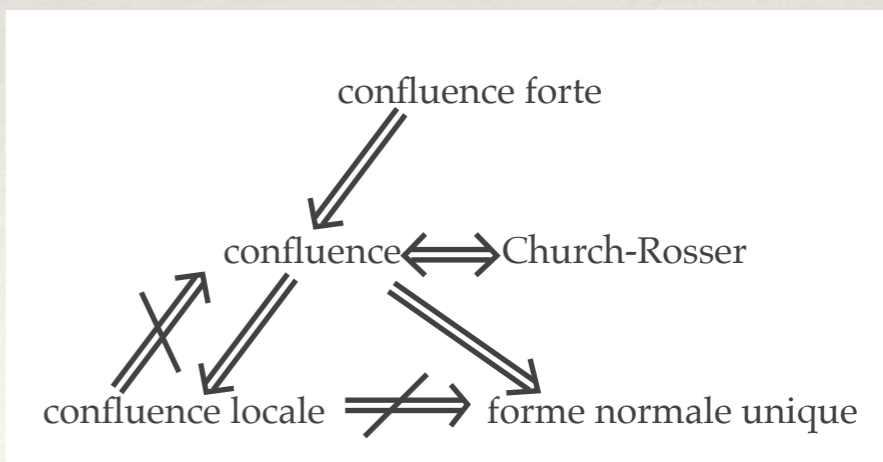
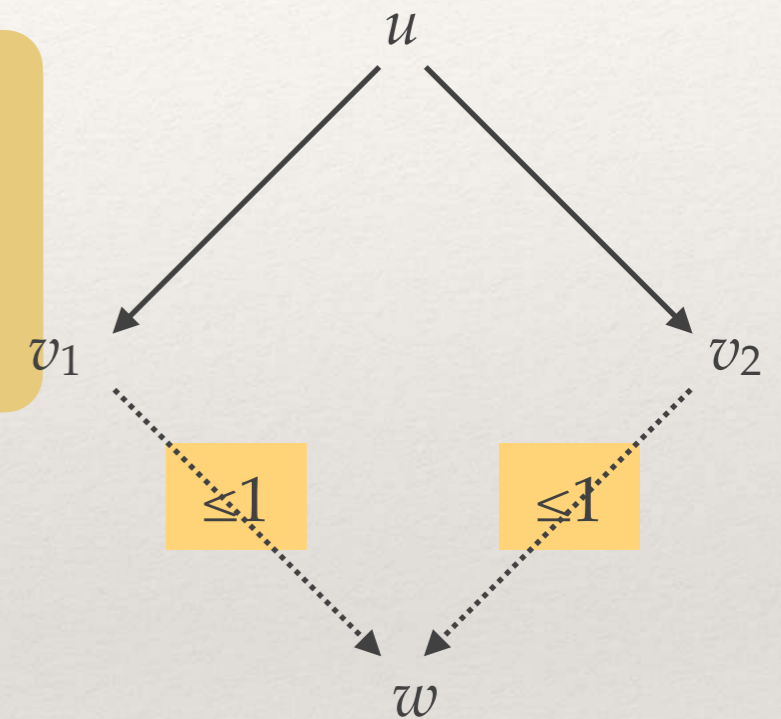
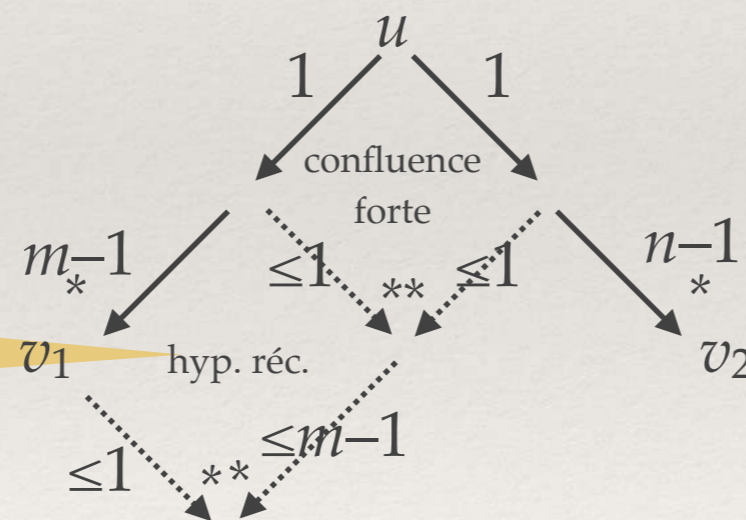
Confluence forte

❖ \rightarrow est fortement confluente ssi:

❖ **Lemme.** La confluence forte implique la confluence.

❖ Si $m \geq 1$ et $n \geq 1$:

s'applique car
 $(m-1)+1 = m$
 $< m+n$



Confluence forte

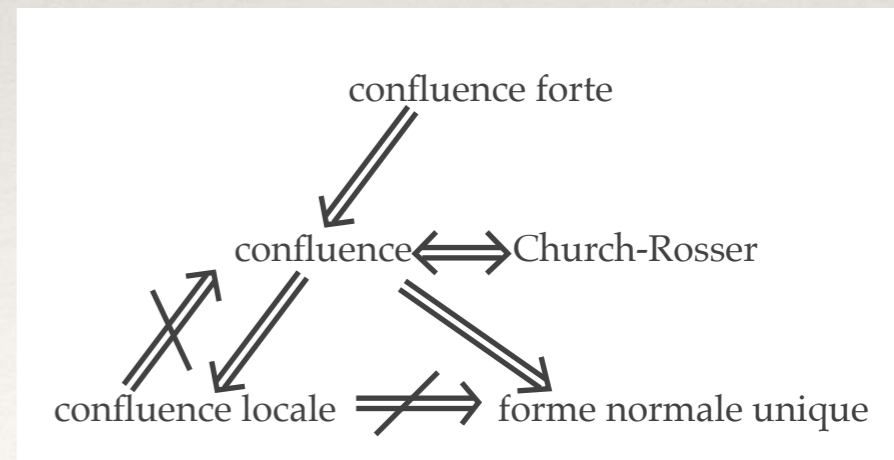
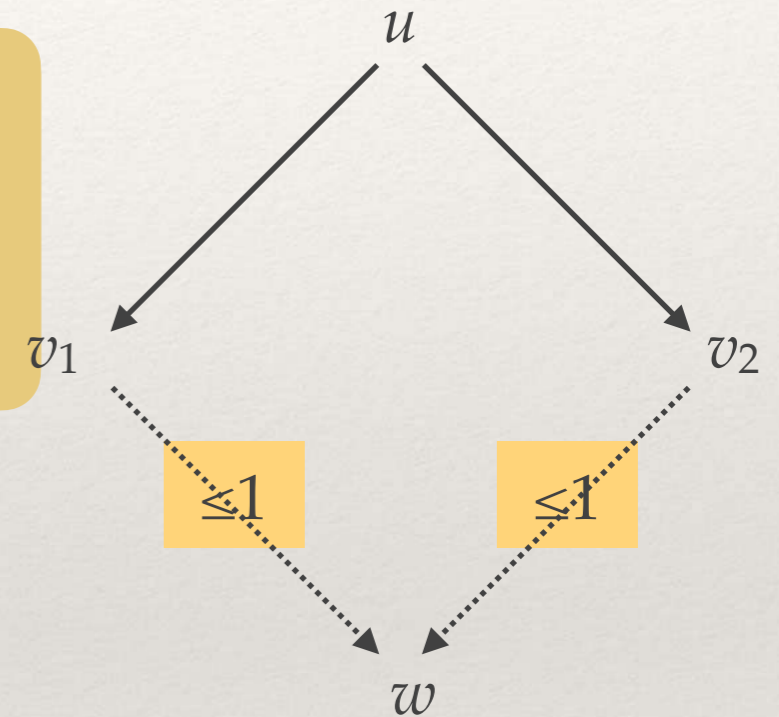
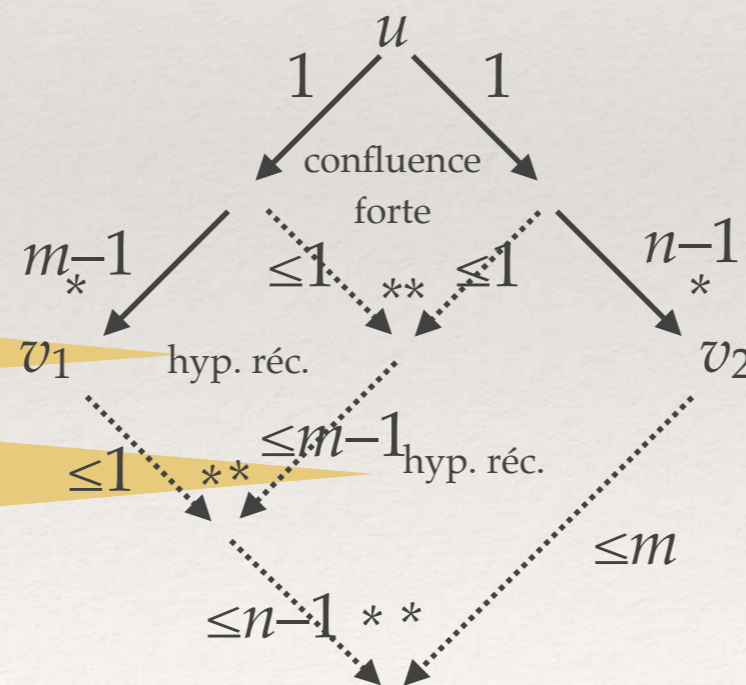
❖ \rightarrow est fortement confluente ssi:

❖ **Lemme.** La confluence forte implique la confluence.

❖ Si $m \geq 1$ et $n \geq 1$:

s'applique car
 $(m-1)+1 = m$
 $< m+n$

s'applique car
 $(m-1+1)+(n-1)$
 $= m+n-1$
 $< m+n$



Le λ -calcul est-il fortement confluant?

❖ **Non...** 😞

❖ Imaginons

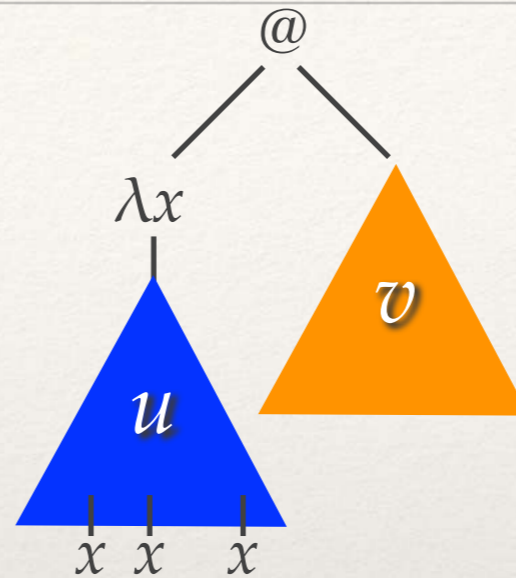
que  v \rightarrow  w . Alors:

Le λ -calcul est-il fortement confluente?

❖ **Non...** 😞

❖ Imaginons

que $v \rightarrow w$. Alors:

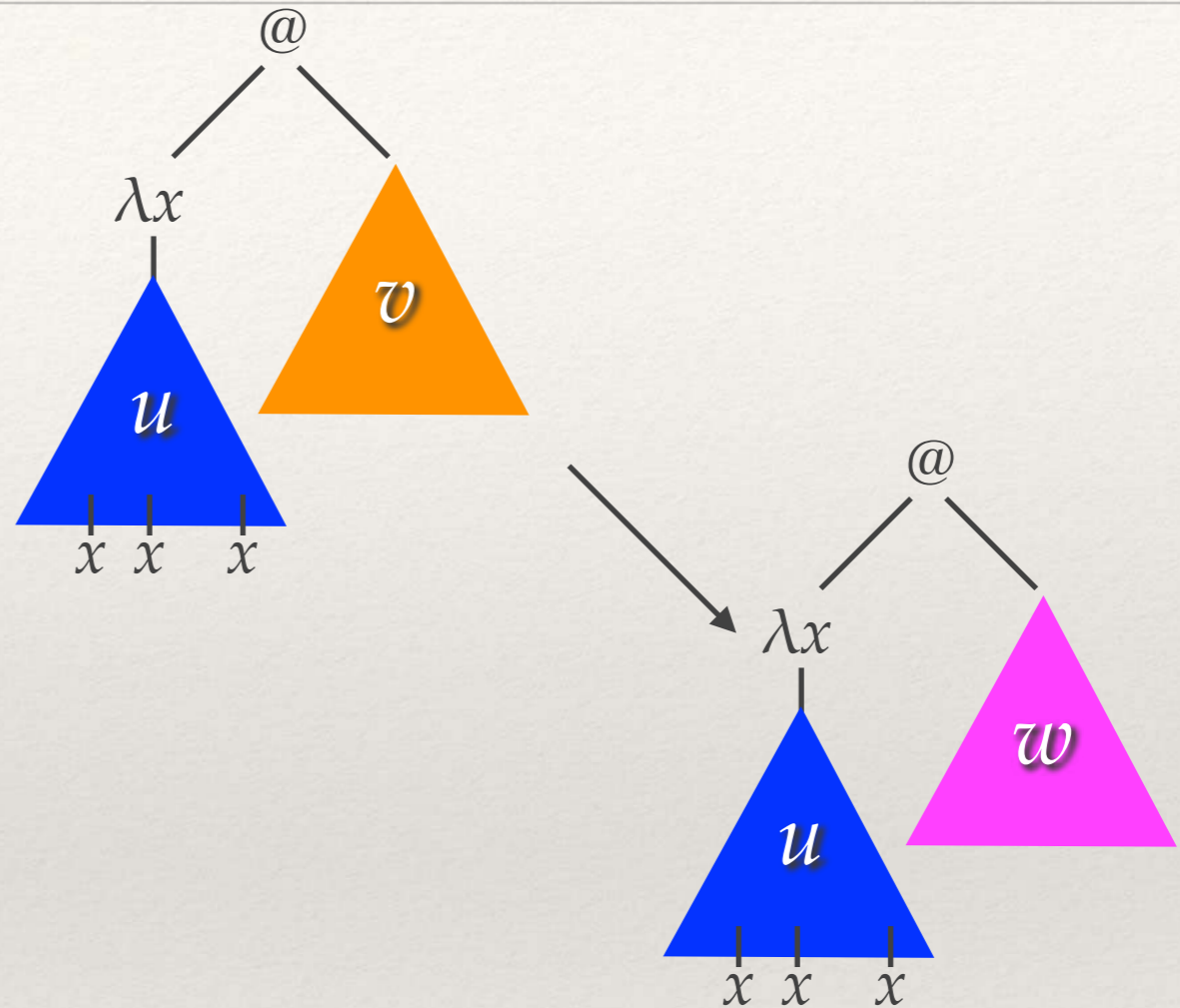


Le λ -calcul est-il fortement confluente?

❖ Non... 😞

❖ Imaginons

que $v \rightarrow w$. Alors:



Le λ -calcul est-il fortement confluente?

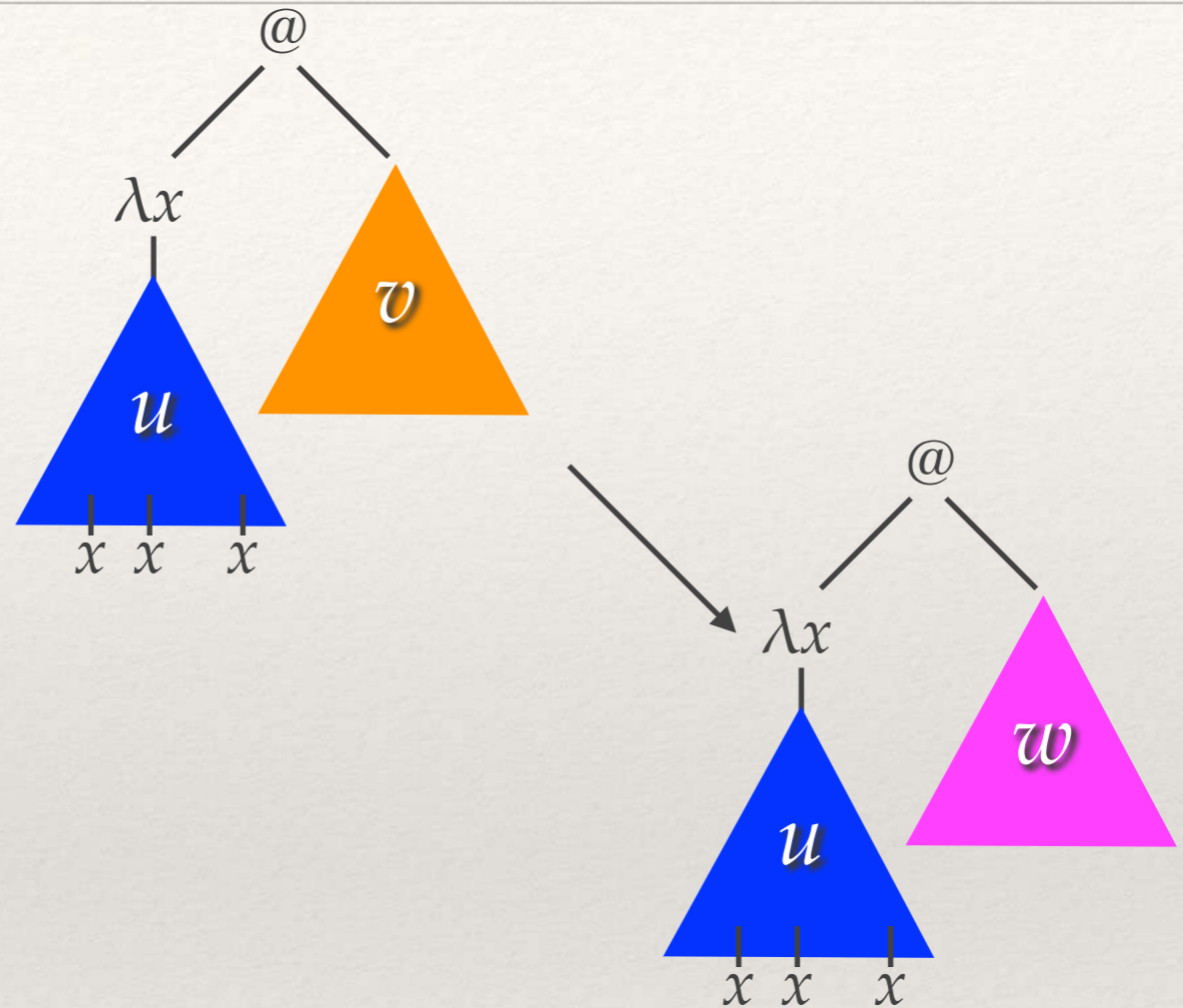
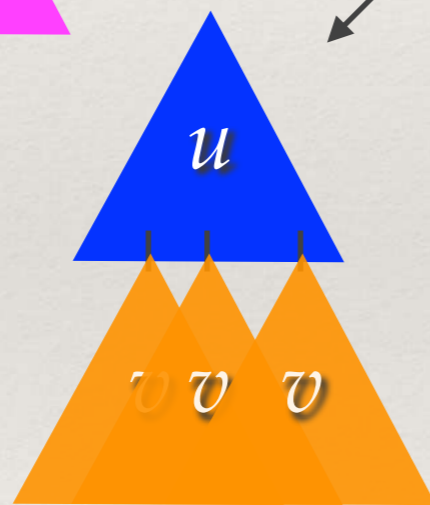
❖ Non... 😞

❖ Imaginons

que



Alors:

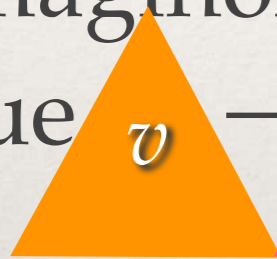


Le λ -calcul est-il fortement confluant?

❖ Non... 😞

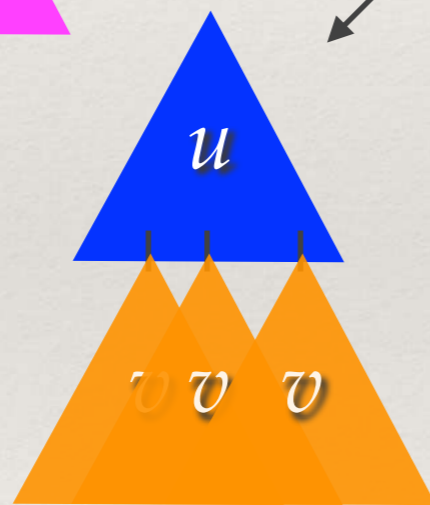
❖ Imaginons

que



w

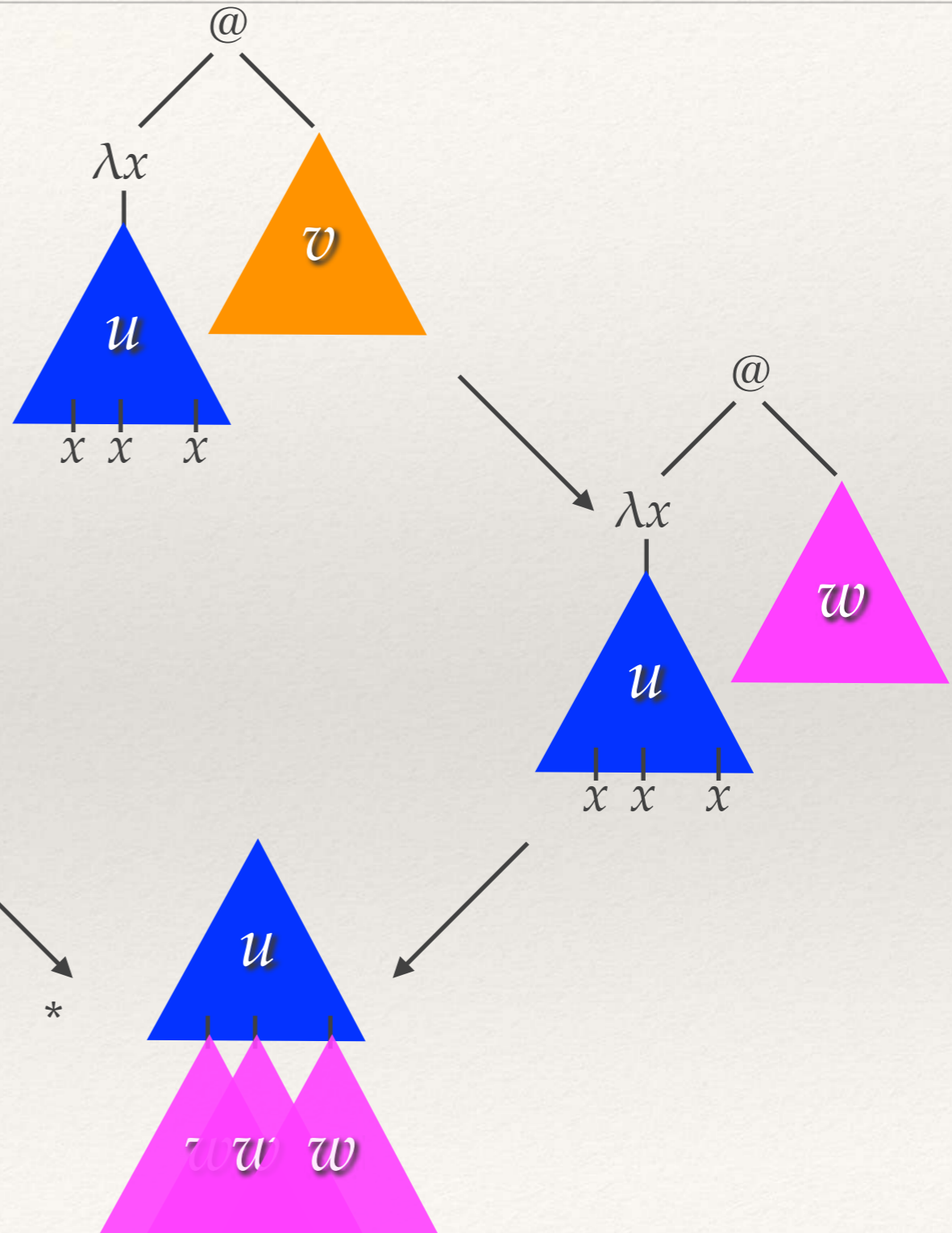
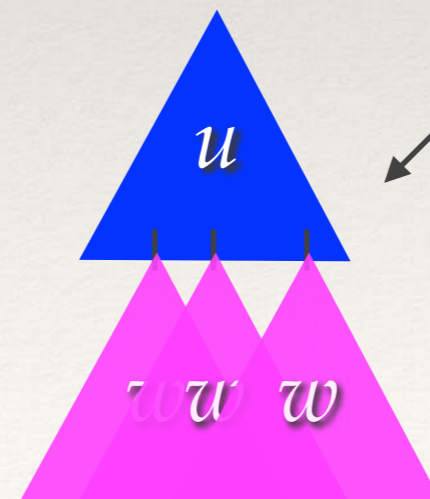
Alors:



3 étapes

(en général,
autant que
d'occurrences
de x dans u)

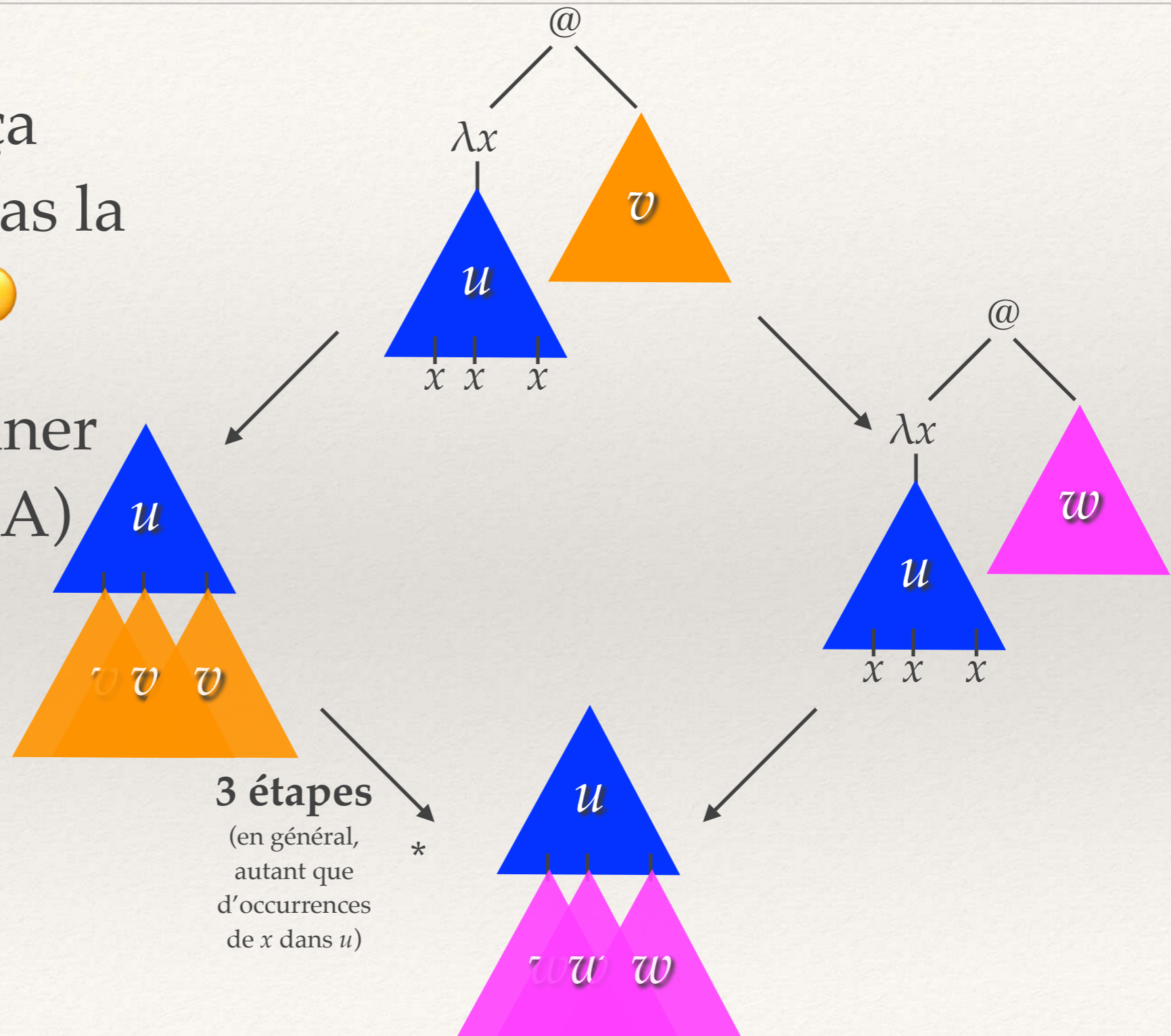
*



Le λ -calcul est-il localement confluent?

❖ **Oui...** mais ça n'implique pas la confluence 😞

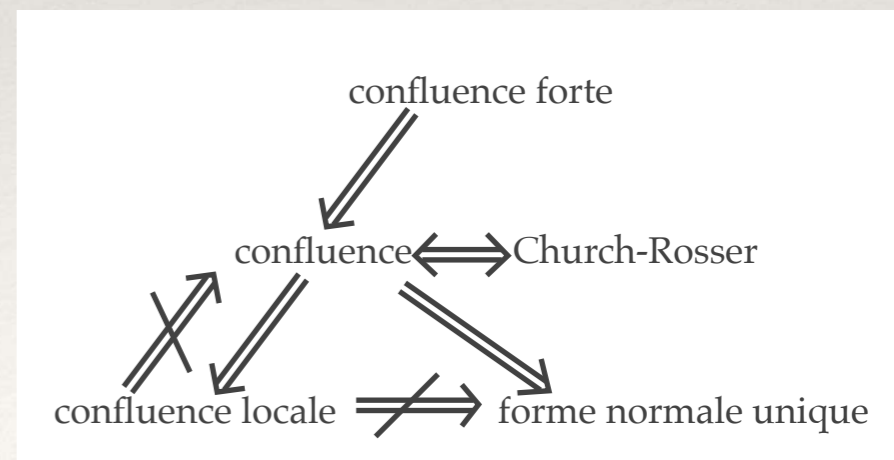
❖ 3 cas à examiner (voir annexe A)



Le lemme de Newman

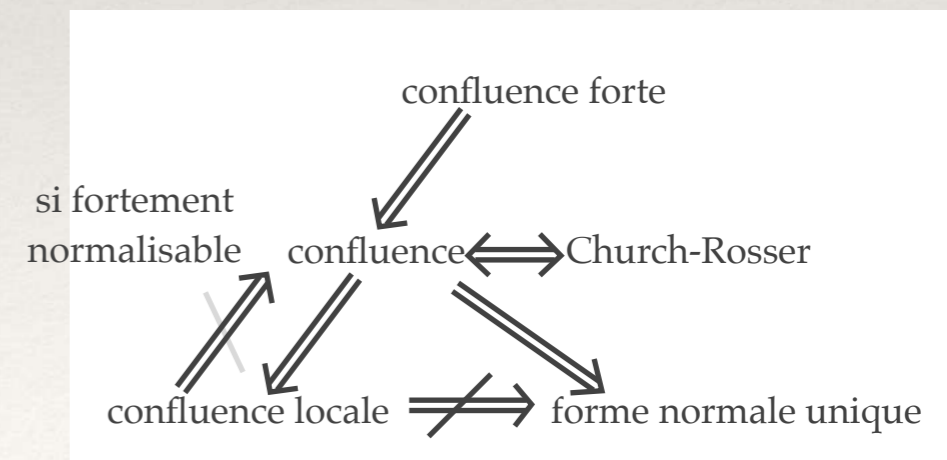
Le lemme de Newman

- ❖ **Lemme (Newman 1941).** Toute relation localement confluente et fortement normalisable est confluente.
- ❖ Preuve(s): transparents suivants.



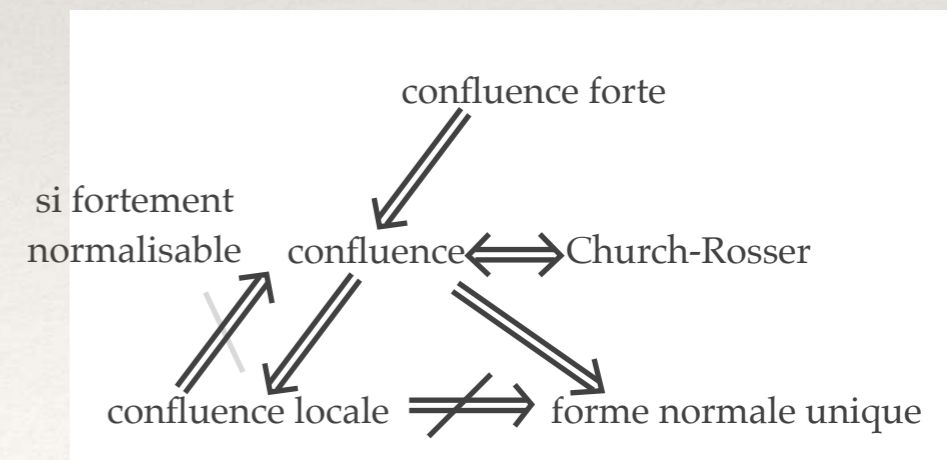
Le lemme de Newman

- ❖ **Lemme (Newman 1941).** Toute relation localement confluente et fortement normalisable est confluente.
- ❖ Preuve(s): transparents suivants.



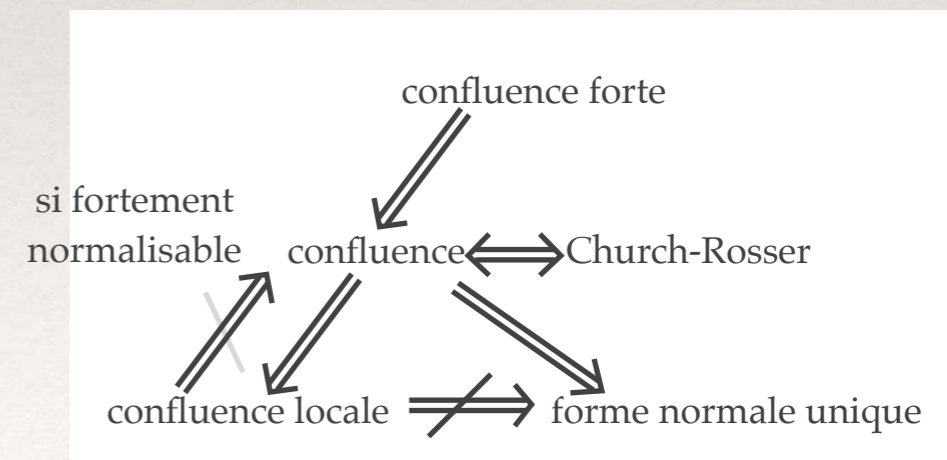
Le lemme de Newman

- ❖ **Lemme (Newman 1941).** Toute relation localement confluente et fortement normalisable est confluente.
- ❖ Preuve(s): transparents suivants.
- ❖ **Note 1:** le contre-ex. de Curry n'est **pas** fortement normalisable



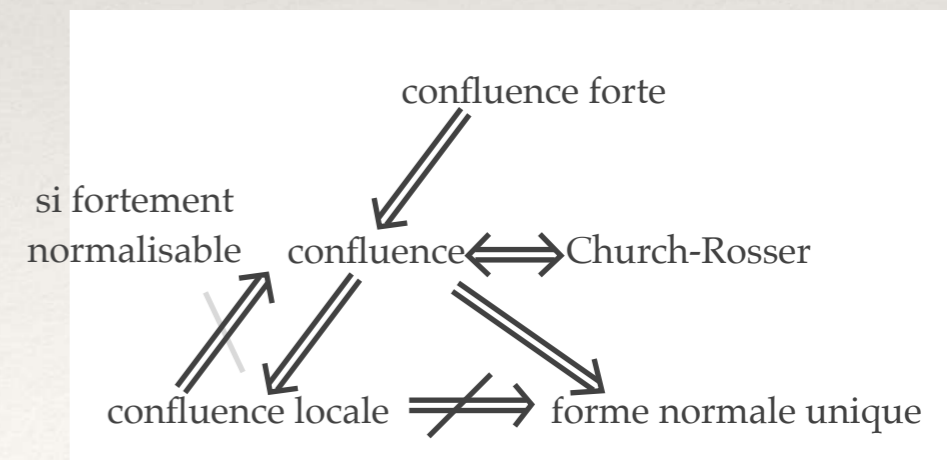
Le lemme de Newman

- ❖ **Lemme (Newman 1941).** Toute relation localement confluente et fortement normalisable est confluente.
- ❖ Preuve(s): transparents suivants.
- ❖ **Note 1:** le contre-ex. de Curry n'est **pas** fortement normalisable
- ❖ **Note 2:** la normalisabilité (faible) ne suffit pas (cf. Curry)



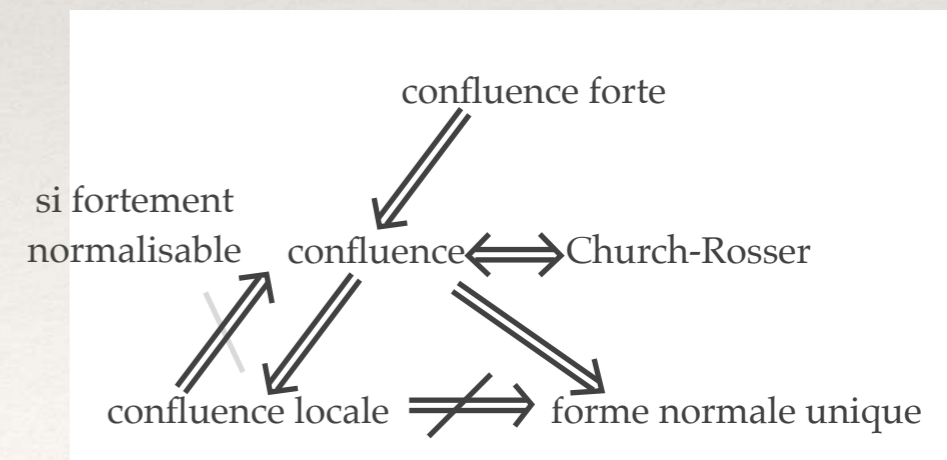
Le lemme de Newman

- ❖ **Lemme (Newman 1941).** Toute relation localement confluente et fortement normalisable est confluente.
- ❖ Preuve(s): transparents suivants.
- ❖ **Note 1:** le contre-ex. de Curry n'est **pas** fortement normalisable
- ❖ **Note 2:** la normalisabilité (faible) ne suffit pas (cf. Curry)
- ❖ **Note 3:** ne s'applique toujours pas au λ -calcul, qui n'est pas fortement normalisable...
mais sera utile quand même!



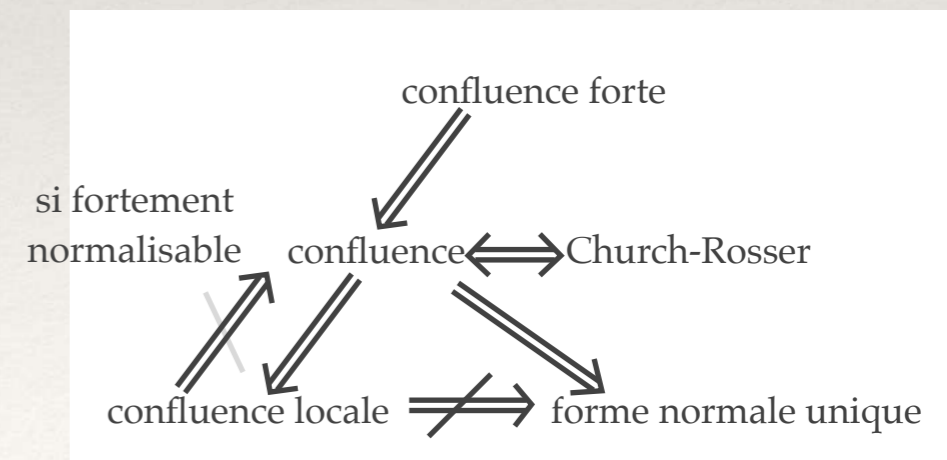
Le lemme de Newman

- ❖ **Lemme (Newman 1941).** Toute relation localement confluente et fortement normalisable est confluente.
- ❖ On va en donner une preuve sous l'hypothèse supplémentaire que \rightarrow est à **branchement fini**:
pour tout u , $\{v \mid u \rightarrow v\}$ est fini
(c'est clairement le cas en λ -calcul, et donc ça nous suffira)



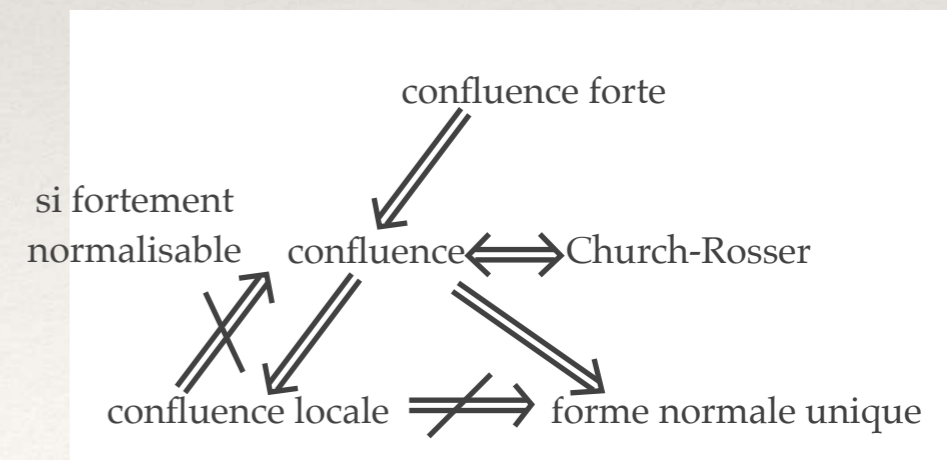
Le lemme de Newman

- ❖ **Lemme (Newman 1941).** Toute relation localement confluente et fortement normalisable est confluente.
- ❖ On va en donner une preuve sous l'hypothèse supplémentaire que \rightarrow est à **branchement fini**:
pour tout u , $\{v \mid u \rightarrow v\}$ est fini
(c'est clairement le cas en λ -calcul, et donc ça nous suffira)
- ❖ Alors pour tout u , $v(u) \stackrel{\text{def}}{=} \text{longueur max. d'une réduction partant de } u$ existe, par le lemme de König (détails en annexe B)



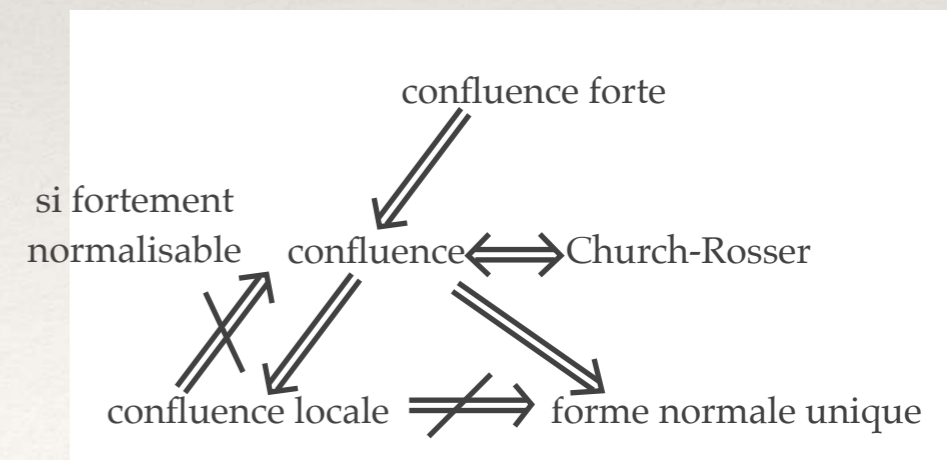
Le lemme de Newman: 1ère preuve

- ❖ **Lemme (Newman 1941).** Toute relation localement confluente et fortement normalisable et à branchement fini est confluente.



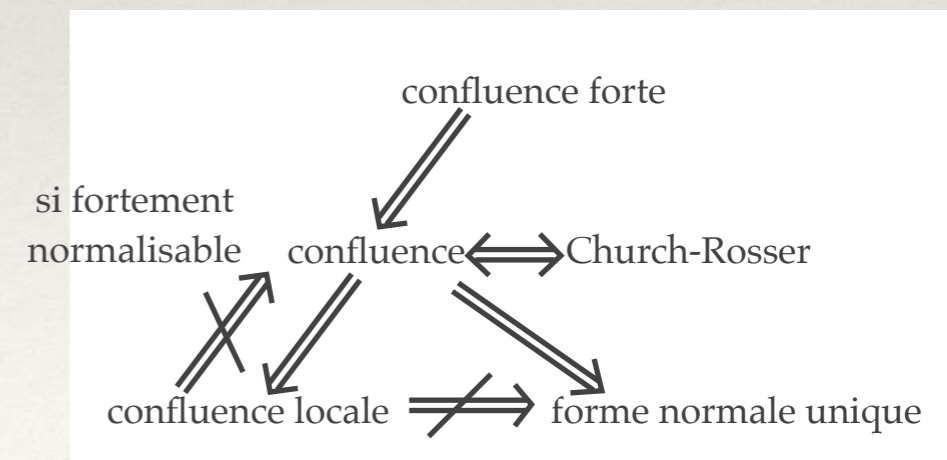
Le lemme de Newman: 1ère preuve

- ❖ **Lemme (Newman 1941).** Toute relation localement confluente et fortement normalisable et à branchement fini est confluente.
- ❖ On montre que si $v_1 \leftarrow^* u \rightarrow^* v_2$ alors v_1 et v_2 ont un réduct commun par récurrence sur $v(u)$



Le lemme de Newman: 1ère preuve

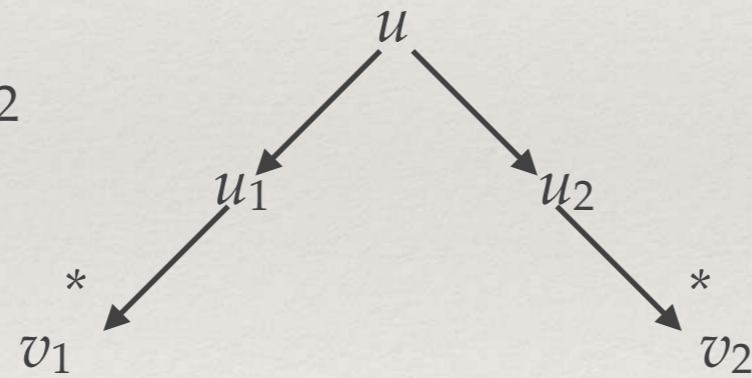
- ❖ **Lemme (Newman 1941).** Toute relation localement confluente et fortement normalisable et à branchement fini est confluente.
- ❖ On montre que si $v_1 \leftarrow^* u \rightarrow^* v_2$ alors v_1 et v_2 ont un réduct commun par récurrence sur $v(u)$
- ❖ Les cas $u=v_1$ et $u=v_2$ sont comme avant



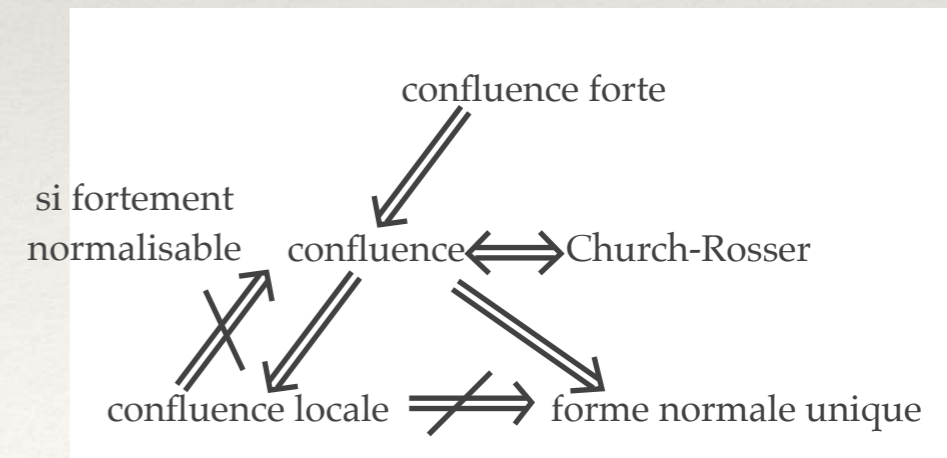
Le lemme de Newman: 1ère preuve

- ❖ **Lemme (Newman 1941).** Toute relation localement confluente et fortement normalisable et à branchement fini est confluente.
- ❖ On montre que si $v_1 \leftarrow^* u \rightarrow^* v_2$ alors v_1 et v_2 ont un réduit commun par récurrence sur $v(u)$

- ❖ Les cas $u=v_1$ et $u=v_2$ sont comme avant



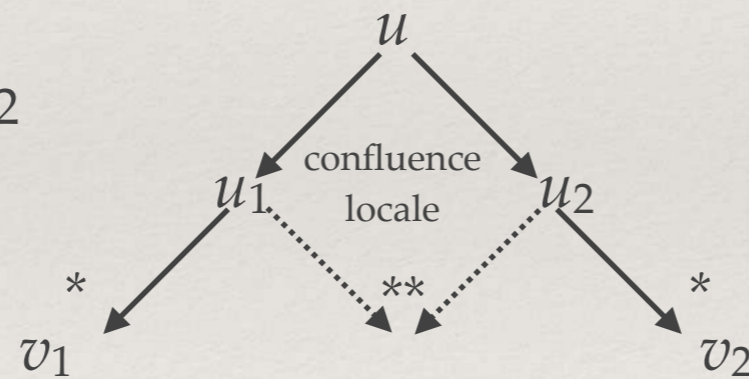
- ❖ Sinon:



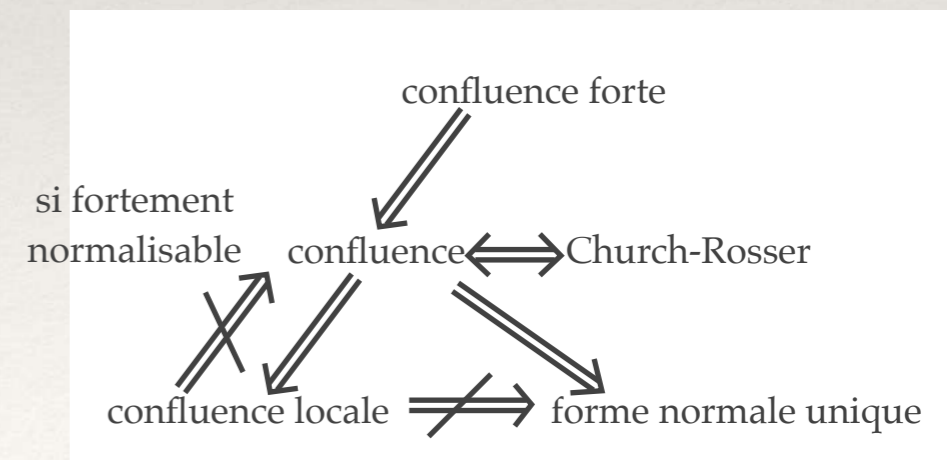
Le lemme de Newman: 1ère preuve

- ❖ **Lemme (Newman 1941).** Toute relation localement confluente et fortement normalisable et à branchement fini est confluente.
- ❖ On montre que si $v_1 \leftarrow^* u \rightarrow^* v_2$ alors v_1 et v_2 ont un réduit commun par récurrence sur $v(u)$

- ❖ Les cas $u=v_1$ et $u=v_2$ sont comme avant



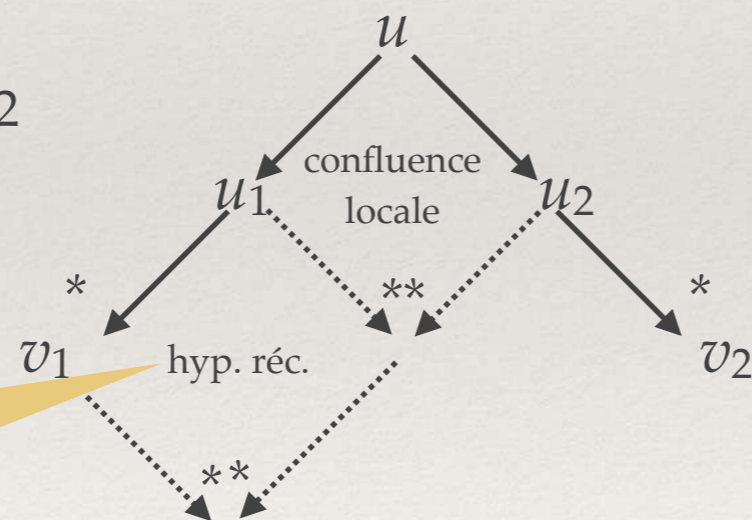
- ❖ Sinon:



Le lemme de Newman: 1ère preuve

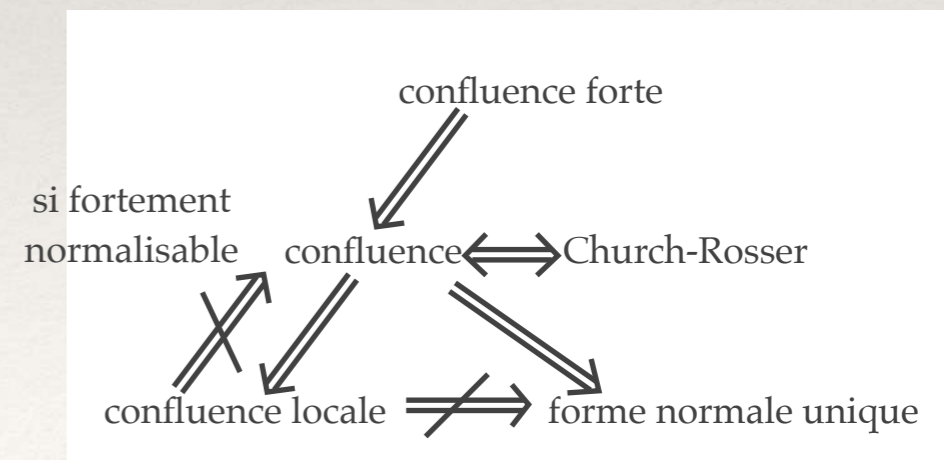
- ❖ **Lemme (Newman 1941).** Toute relation localement confluente et fortement normalisable est confluente.
- ❖ On montre que si $v_1 \leftarrow^* u \rightarrow^* v_2$ alors v_1 et v_2 ont un réduct commun par récurrence sur $v(u)$

- ❖ Les cas $u=v_1$ et $u=v_2$ sont comme avant



- ❖ Sinon:

s'applique car $v(u) > v(u_1)$



Le lemme de Newman: 1ère preuve

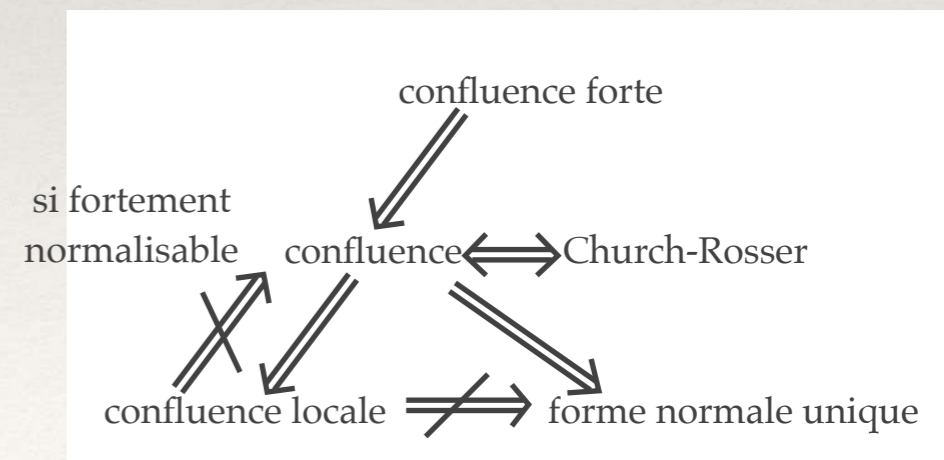
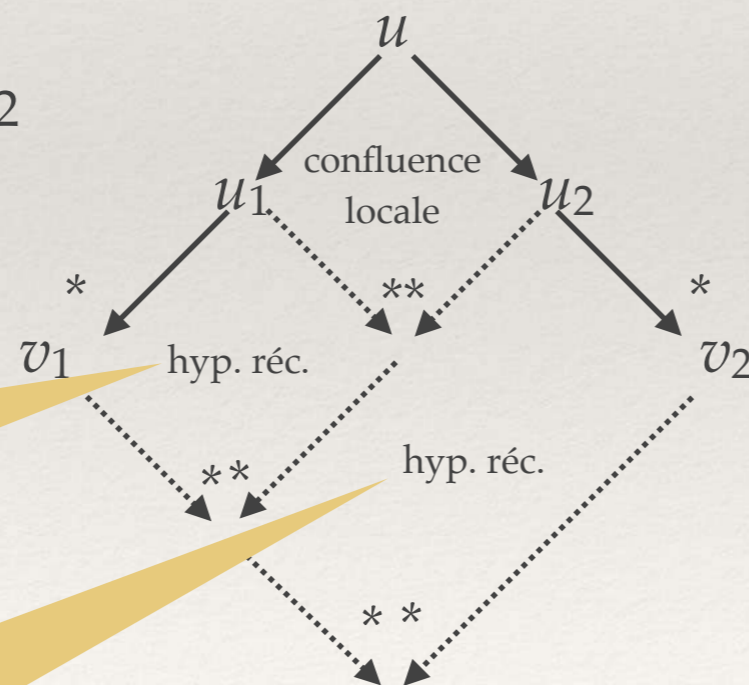
- ❖ **Lemme (Newman 1941).** Toute relation localement confluente et fortement normalisable est confluente.
- ❖ On montre que si $v_1 \leftarrow^* u \rightarrow^* v_2$ alors v_1 et v_2 ont un réduit commun par récurrence sur $v(u)$

- ❖ Les cas $u=v_1$ et $u=v_2$ sont comme avant

- ❖ Sinon:

s'applique car $v(u) > v(u_1)$

s'applique car $v(u) > v(u_2)$



Le λ -calcul est confluent

Réductions parallèles

- ❖ La preuve standard de confluence du λ -calcul
- ❖ On introduit une relation \Rightarrow de **réduction parallèle** qui intuitivement autorise à réduire plusieurs redexes, à condition qu'aucun n'en contienne un autre
- ❖ On montre que \Rightarrow est **fortement confluente**
donc confluente
- ❖ Enfin, $\rightarrow \subseteq \Rightarrow \subseteq \rightarrow^*$
donc \rightarrow sera confluente.

Réductions parallèles

$$\frac{}{u \Rightarrow u} \quad (0)$$
$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$
$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$
$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

- ❖ Pour simplifier, raisonnons à α -équivalence près:
les règles suivantes sont donc inutiles

$$\frac{u =_{\alpha} v \quad v \rightarrow v'}{u \rightarrow v'} \quad \frac{u \rightarrow u' \quad u' =_{\alpha} v'}{u \rightarrow v'}$$

Réductions parallèles

0 réduction parallèle
(nouvelle règle)

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

- ❖ Pour simplifier, raisonnons à α -équivalence près:
les règles suivantes sont donc inutiles

$$\frac{u =_{\alpha} v \quad v \rightarrow v'}{u \rightarrow v'} \quad \frac{u \rightarrow u' \quad u' =_{\alpha} v'}{u \rightarrow v'}$$

Réductions parallèles

0 réduction parallèle
(nouvelle règle)

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

On peut réduire
en parallèle
dans u et dans v

- ❖ Pour simplifier, raisonnons à α -équivalence près:
les règles suivantes sont donc inutiles

$$\frac{u =_{\alpha} v \quad v \rightarrow v'}{u \rightarrow v'} \quad \frac{u \rightarrow u' \quad u' =_{\alpha} v'}{u \rightarrow v'}$$

Réductions parallèles

0 réduction parallèle
(nouvelle règle)

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

On peut réduire
en parallèle
dans u et dans v

- ❖ Pour simplifier, raisonnons à α -équivalence près:
les règles suivantes sont donc inutiles

$$\frac{u =_{\alpha} v \quad v \rightarrow v'}{u \rightarrow v'} \quad \frac{u \rightarrow u' \quad u' =_{\alpha} v'}{u \rightarrow v'}$$

- ❖ **Proposition.** \Rightarrow est **fortement** confluente.
- ❖ C'est un exercice standard (voir annexe B).
- ❖ Donc \Rightarrow est **confluente**.



$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

❖ **Lemme** ($\rightarrow \subseteq \Rightarrow$). Si $u \rightarrow v$ alors $u \Rightarrow v$.
 (Récurrence sur la profondeur du rédex contracté dans u .)



$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

❖ **Lemme** ($\rightarrow \subseteq \Rightarrow$). Si $u \rightarrow v$ alors $u \Rightarrow v$.
 (Récurrence sur la profondeur du rédex contracté dans u .)

❖ **Lemme** ($\Rightarrow \subseteq \rightarrow^*$). Si $u \Rightarrow v$ alors $u \rightarrow^* v$.
 (Récurrence sur la taille de la dérivation de $u \Rightarrow v$.)



$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

❖ **Lemme** ($\rightarrow \subseteq \Rightarrow$). Si $u \rightarrow v$ alors $u \Rightarrow v$.
(Récurrence sur la profondeur du rédex contracté dans u .)

❖ **Lemme** ($\Rightarrow \subseteq \rightarrow^*$). Si $u \Rightarrow v$ alors $u \rightarrow^* v$.
(Récurrence sur la taille de la dérivation de $u \Rightarrow v$.)

❖ **Corl.** $\rightarrow^* = \Rightarrow^*$.

Le λ -calcul est confluent

- ❖ On résume:
 - ❖ \Rightarrow est (fortement) **confluente**
 - ❖ $\rightarrow^* = \Rightarrow^*$
- ❖ Si:

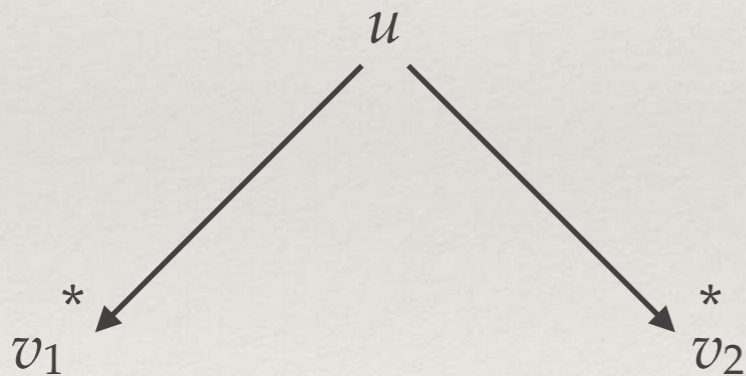
Le λ -calcul est confluente

❖ On résume:

❖ \Rightarrow est (fortement) **confluente**

❖ $\rightarrow^* = \Rightarrow^*$

❖ Si:



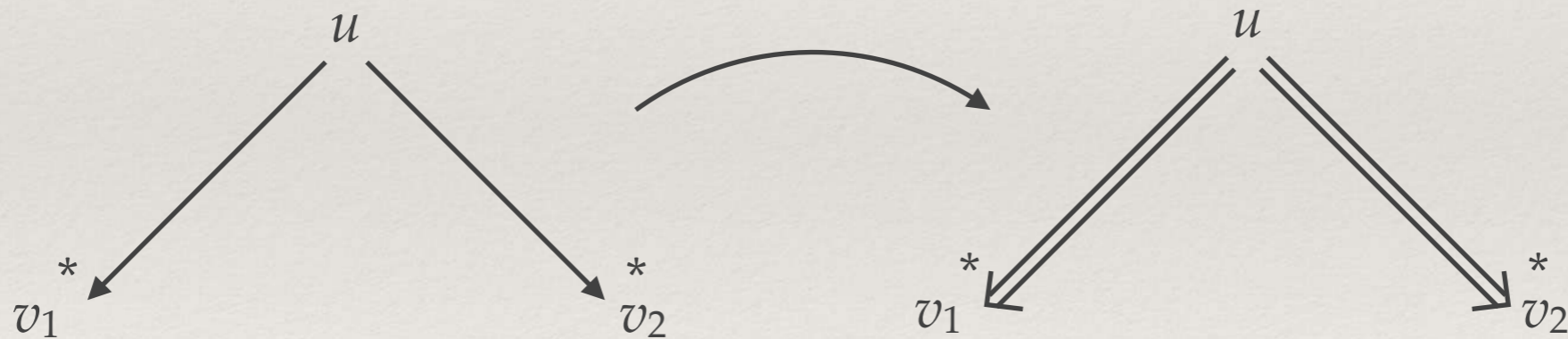
Le λ -calcul est confluent

❖ On résume:

❖ \Rightarrow est (fortement) **confluente**

❖ $\rightarrow^* = \Rightarrow^*$

❖ Si:



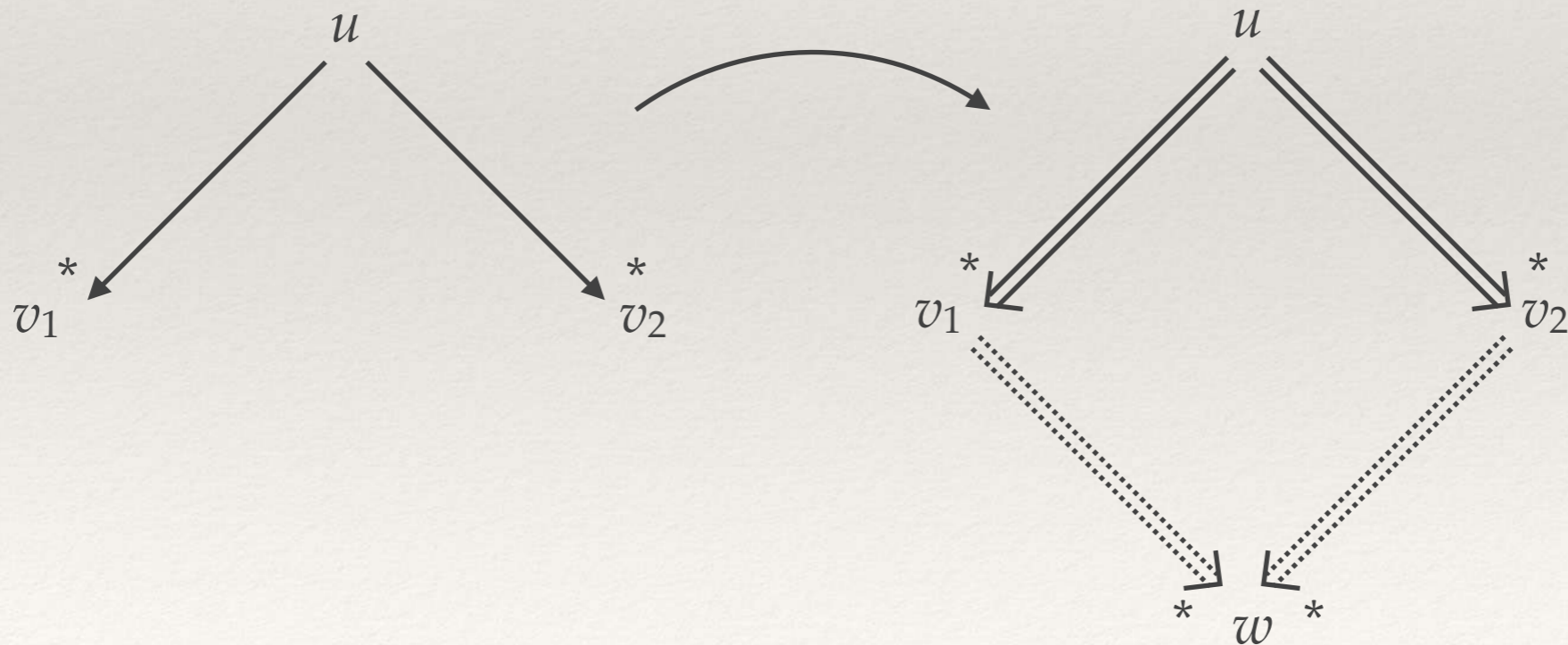
Le λ -calcul est confluent

❖ On résume:

❖ \Rightarrow est (fortement) **confluente**

❖ $\rightarrow^* = \Rightarrow^*$

❖ Si:



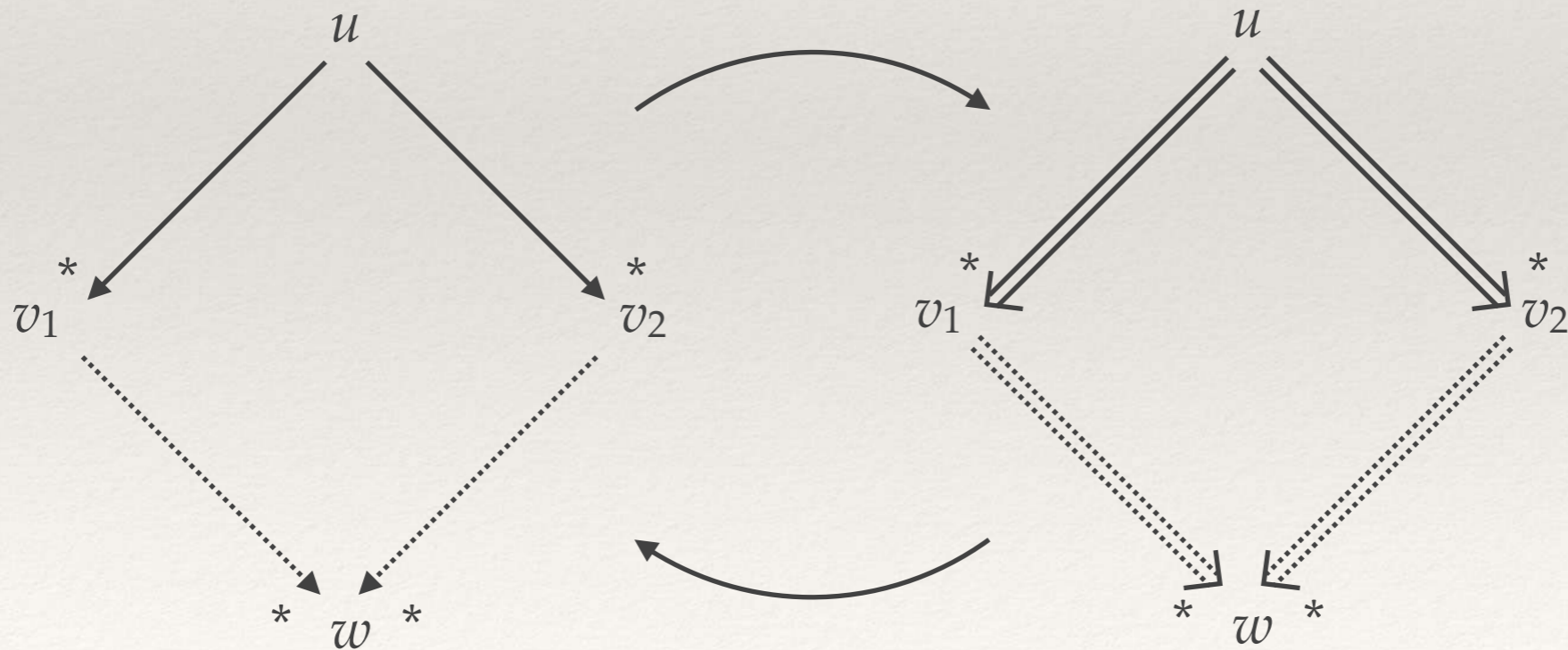
Le λ -calcul est confluent

❖ On résume:

❖ \Rightarrow est (fortement) **confluente**

❖ $\rightarrow^* = \Rightarrow^*$

❖ Si:



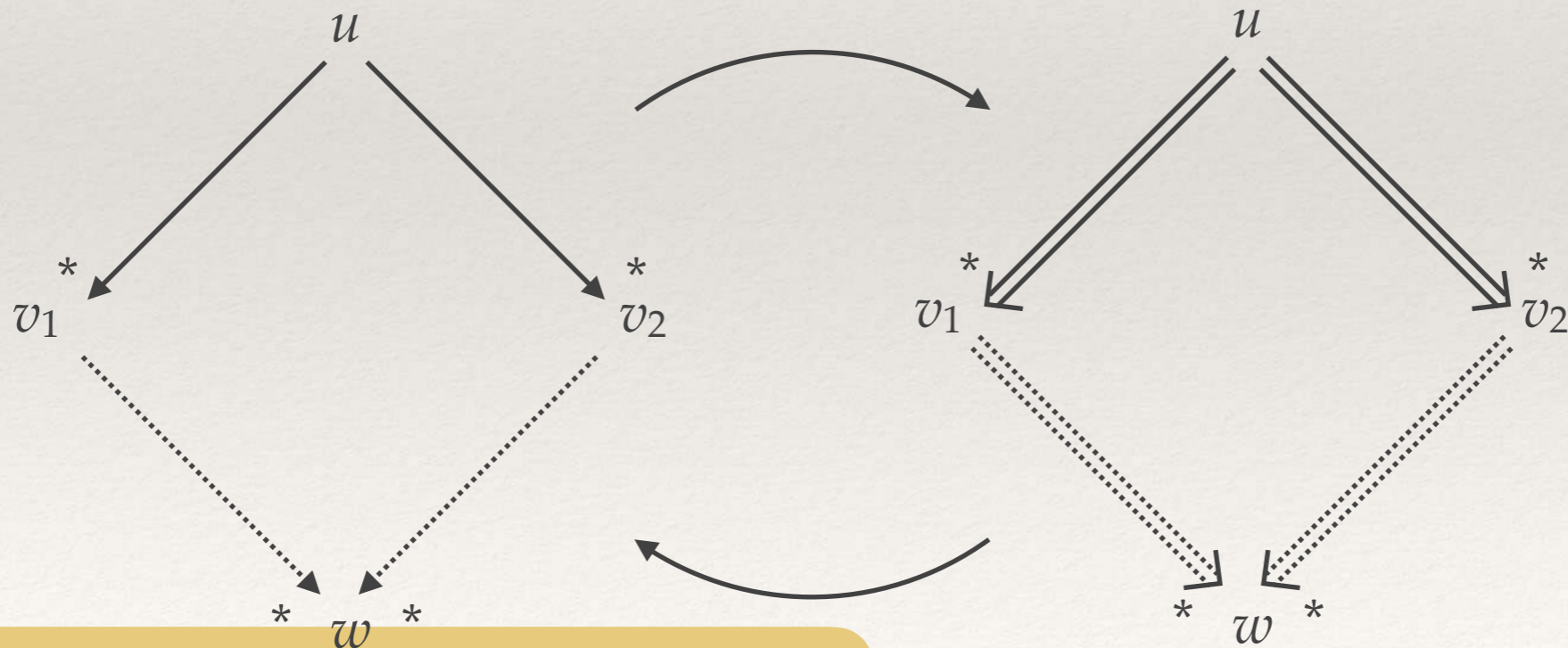
Le λ -calcul est confluent

❖ On résume:

❖ \Rightarrow est (fortement) **confluente**

❖ $\rightarrow^* = \Rightarrow^*$

❖ Si:



Donc le λ -calcul est confluent (pour la β -réduction). \square

Confluence: β n'entraîne pas η

- ❖ Vous vous souvenez de la règle suivante?

$$(\eta) \quad \lambda x . ux \rightarrow u \quad (\text{si } x \text{ pas libre dans } u)$$

et considérer la $\beta\eta$ -réduction $\rightarrow_{\beta\eta}$

- ❖ Elle a l'air raisonnablement mathématiquement (pas informatiquement). Surtout, elle est **indépendante** de β :

Confluence: β n'entraîne pas η

- ❖ Vous vous souvenez de la règle suivante?

$$(\eta) \quad \lambda x . ux \rightarrow u \quad (\text{si } x \text{ pas libre dans } u)$$

et considérer la $\beta\eta$ -réduction $\rightarrow_{\beta\eta}$

- ❖ Elle a l'air raisonnablement mathématiquement (pas informatiquement). Surtout, elle est **indépendante** de β :

- ❖ **Prop.** En général, $\lambda x . ux \neq_{\beta} u$ (même si x pas libre dans u).

Confluence: β n'entraîne pas η

- ❖ Vous vous souvenez de la règle suivante?

$$(\eta) \quad \lambda x . ux \rightarrow u \quad (\text{si } x \text{ pas libre dans } u)$$

et considérer la $\beta\eta$ -réduction $\rightarrow_{\beta\eta}$

- ❖ Elle a l'air raisonnablement mathématiquement (pas informatiquement). Surtout, elle est **indépendante** de β :

- ❖ **Prop.** En général, $\lambda x . ux \neq_{\beta} u$ (même si x pas libre dans u).

- ❖ *Preuve.* On prend $u =$ une variable $\neq x$:

les deux côtés sont en forme β -normale est pas α -équivalents, donc pas β -équivalents

(confluence implique forme normale **unique**). \square

La prochaine fois

- ❖ Pouvoir expressif: fonctions récursives, combinateurs de point fixe
- ❖ Et plus tard: stratégies

Annexe A: le lambda-calcul est localement confluent

Le λ -calcul est-il localement confluente?

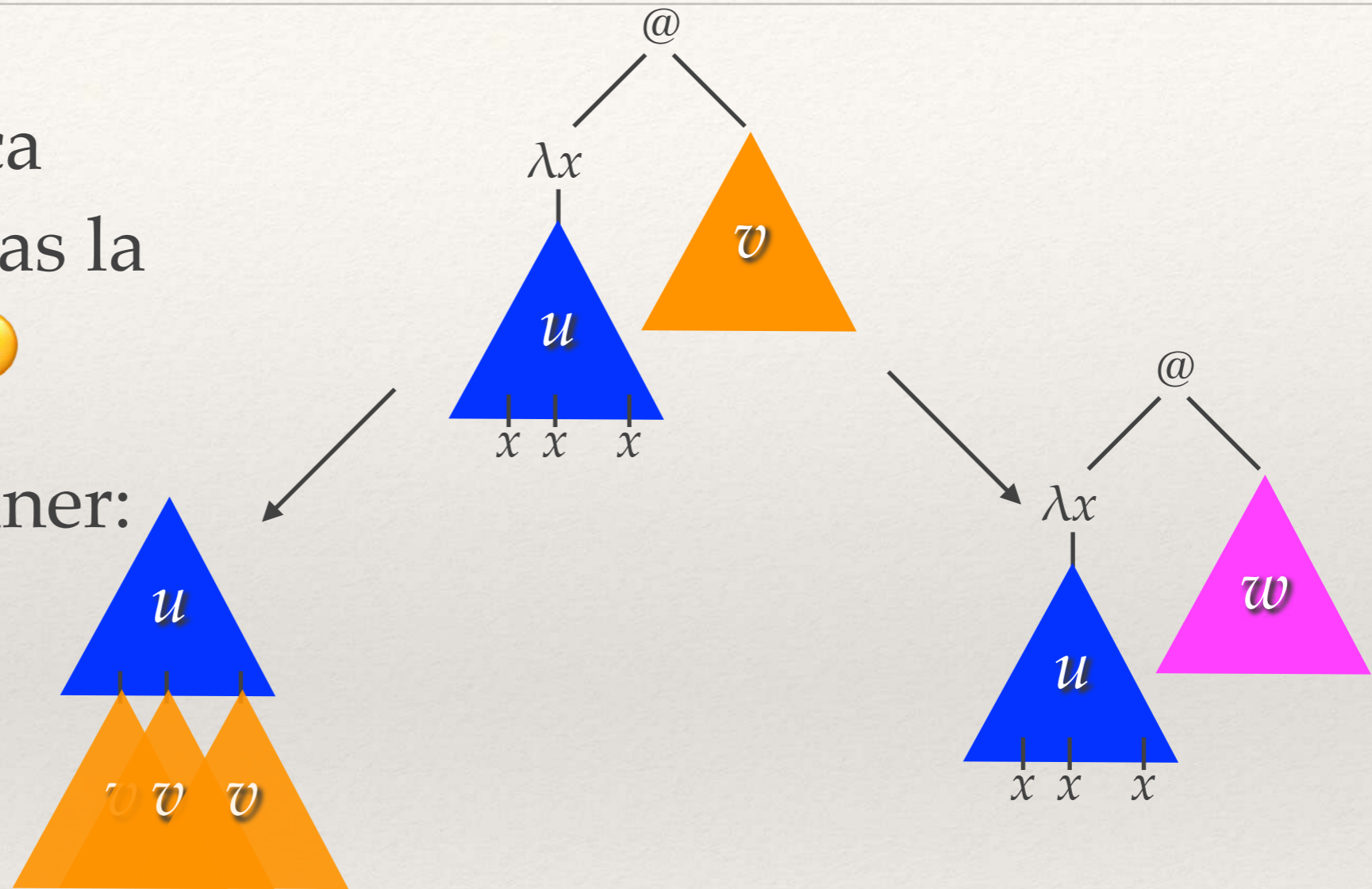
❖ **Oui...** mais ça n'implique pas la confluence 😞

❖ 3 cas à examiner:

❖ **Cas 1:**

$v \rightarrow w,$

v argument d'un rédex



Le λ -calcul est-il localement confluente?

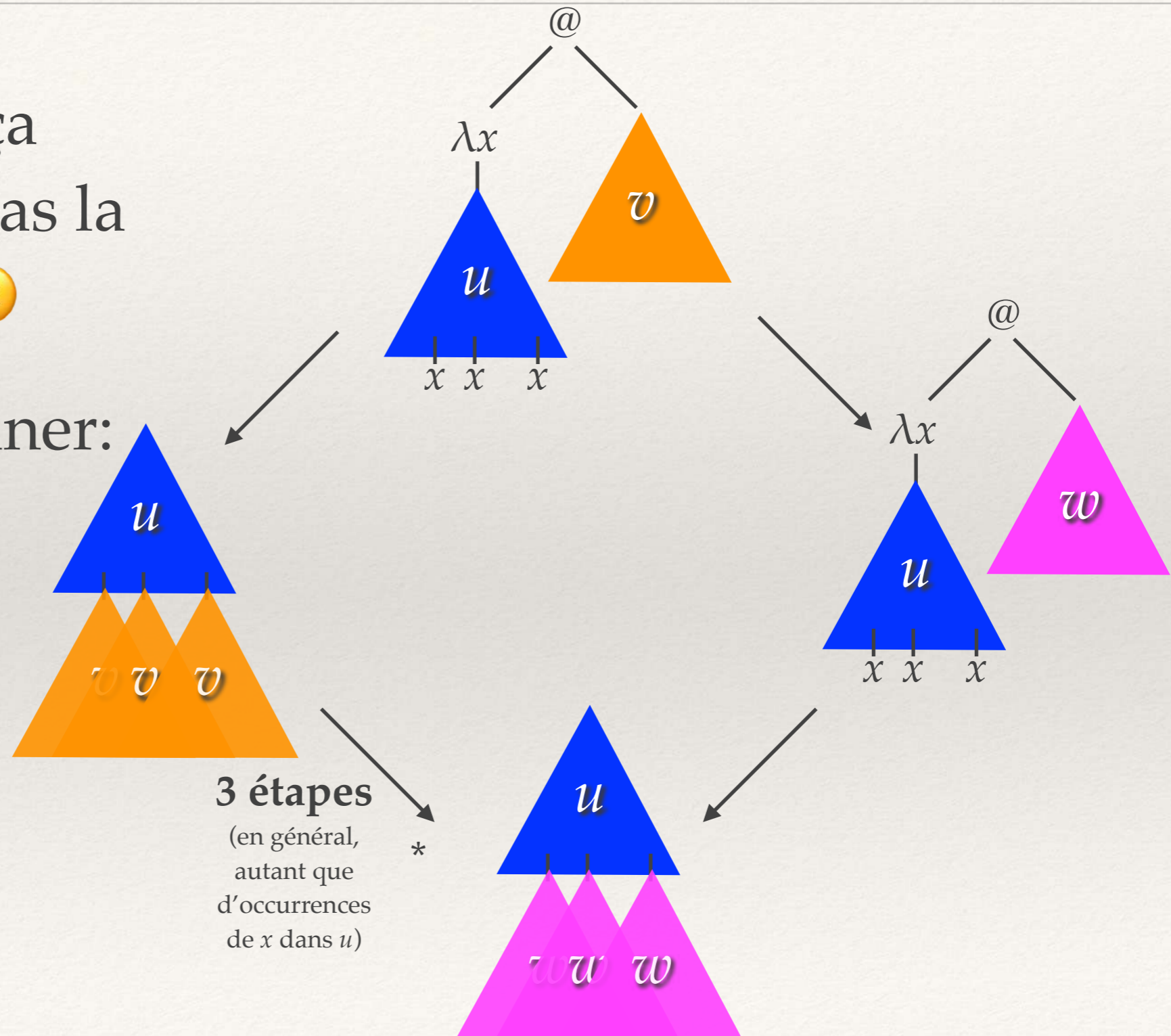
❖ **Oui...** mais ça n'implique pas la confluence 😞

❖ 3 cas à examiner:

❖ **Cas 1:**

$v \rightarrow w$,

v argument d'un rédex



Le λ -calcul est-il localement confluente?

❖ **Oui...** mais ça n'implique pas la confluence 😞

❖ 3 cas à examiner:

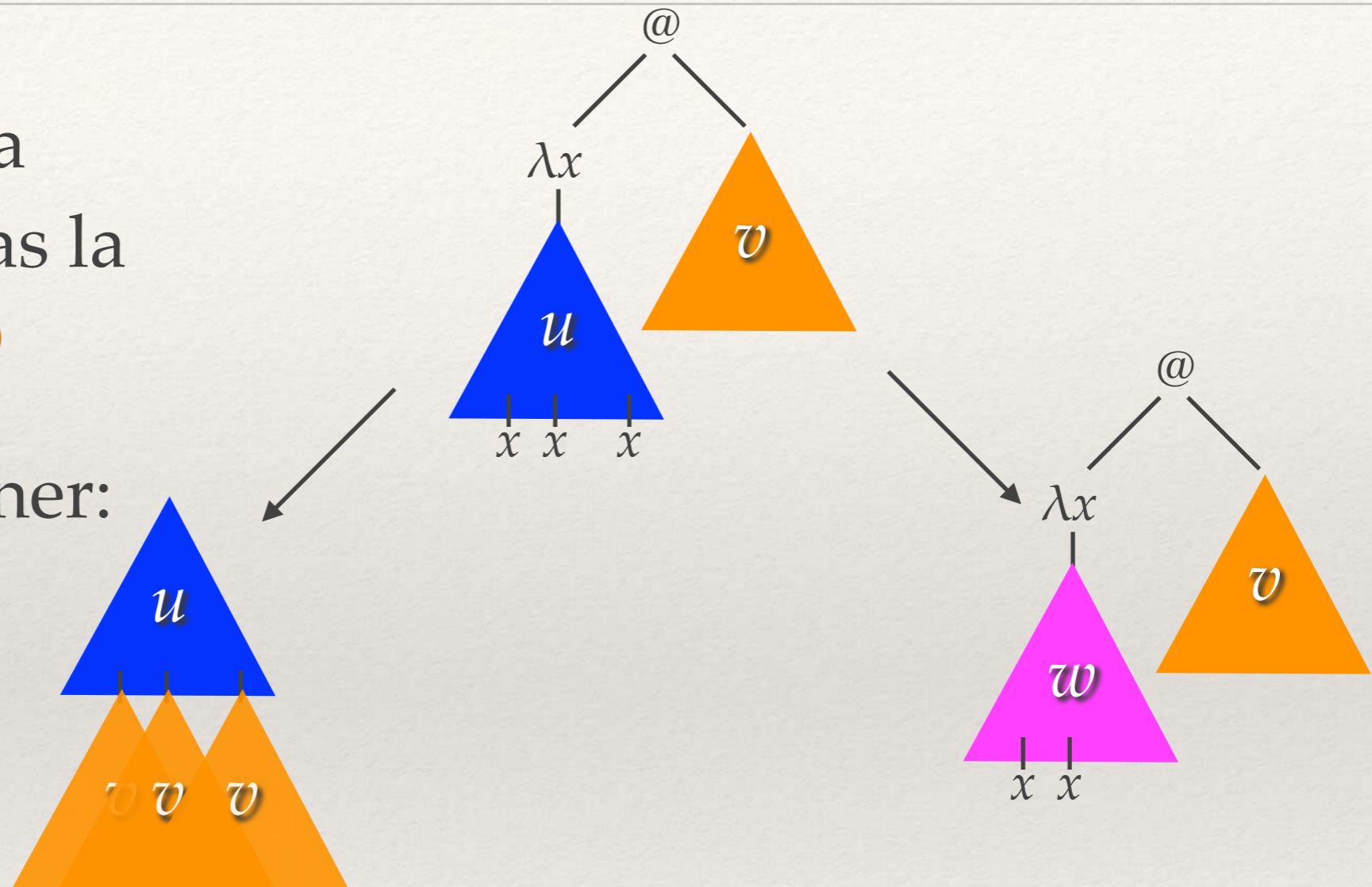
❖ **Cas 2:**

$u \rightarrow w$,

u sous la

λ -abstraction

d'un rédex



Le λ -calcul est-il localement confluent?

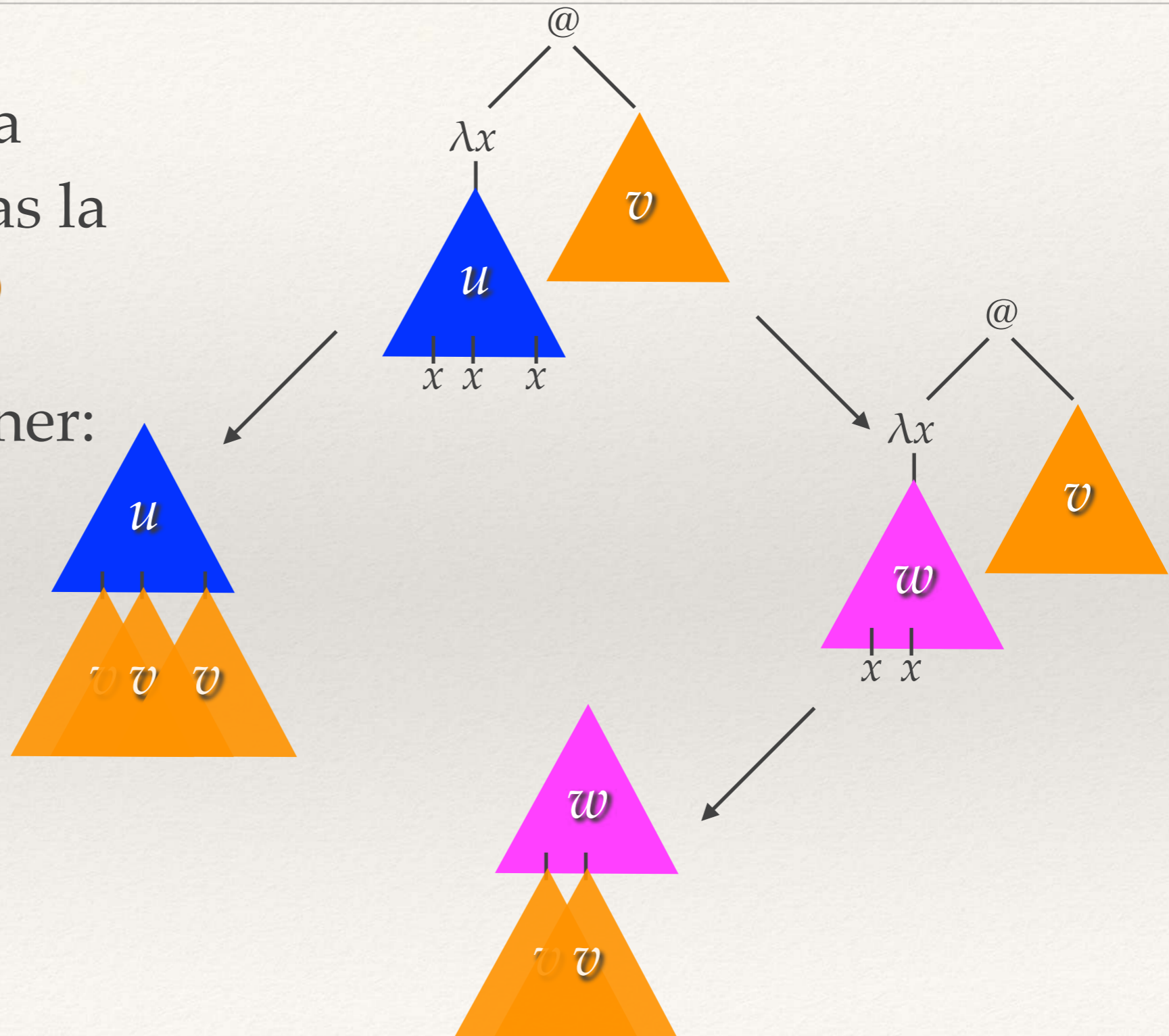
❖ **Oui...** mais ça n'implique pas la confluence 😞

❖ 3 cas à examiner:

❖ **Cas 2:**

$u \rightarrow w$,

u sous la λ -abstraction d'un rédex



Le λ -calcul est-il localement confluent?

❖ **Oui...** mais ça n'implique pas la confluence 😞

❖ 3 cas à examiner:

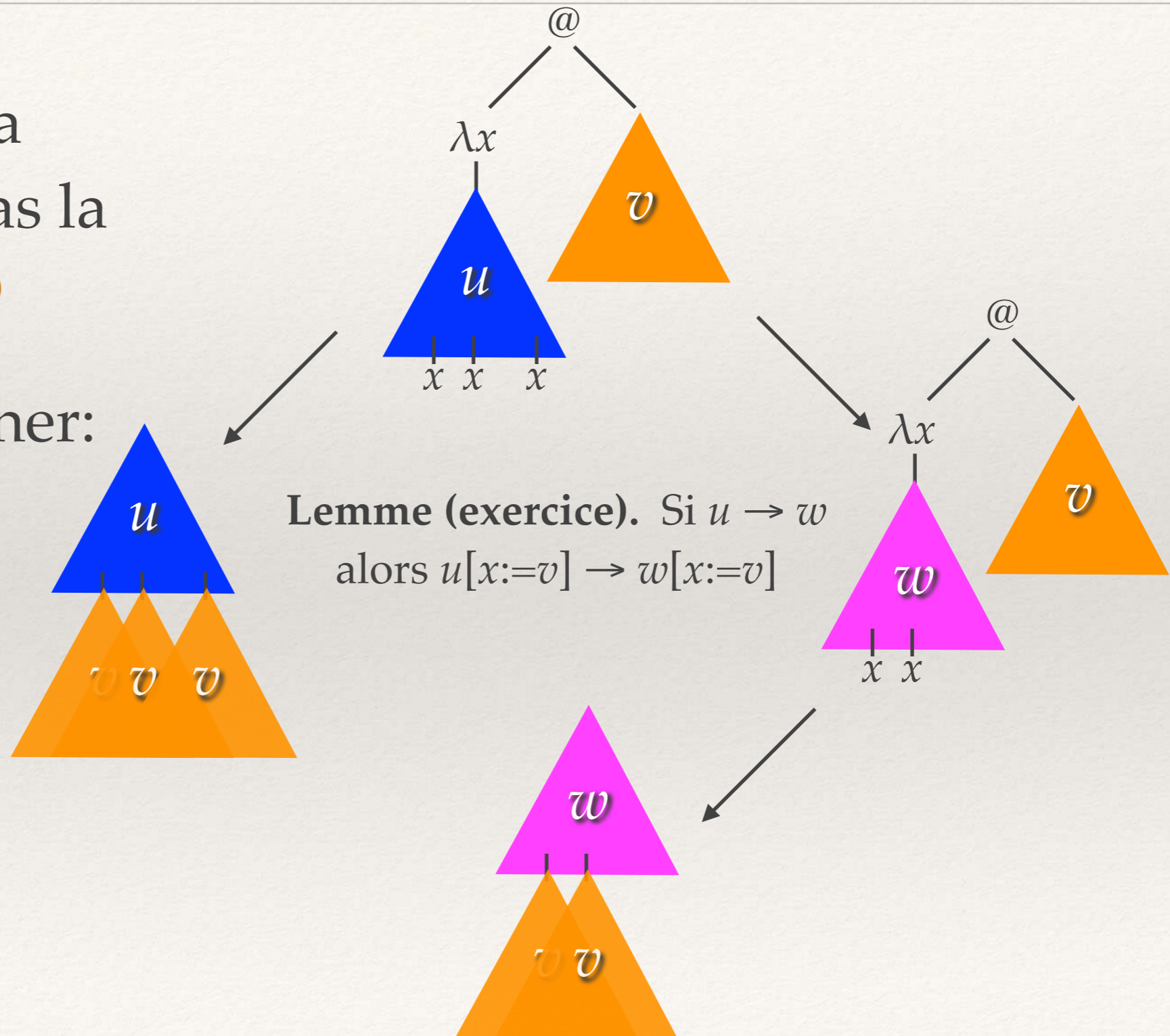
❖ **Cas 2:**

$u \rightarrow w$,

u sous la

λ -abstraction

d'un rédex



Le λ -calcul est-il localement confluente?

❖ **Oui...** mais ça n'implique pas la confluence 😞

❖ 3 cas à examiner:

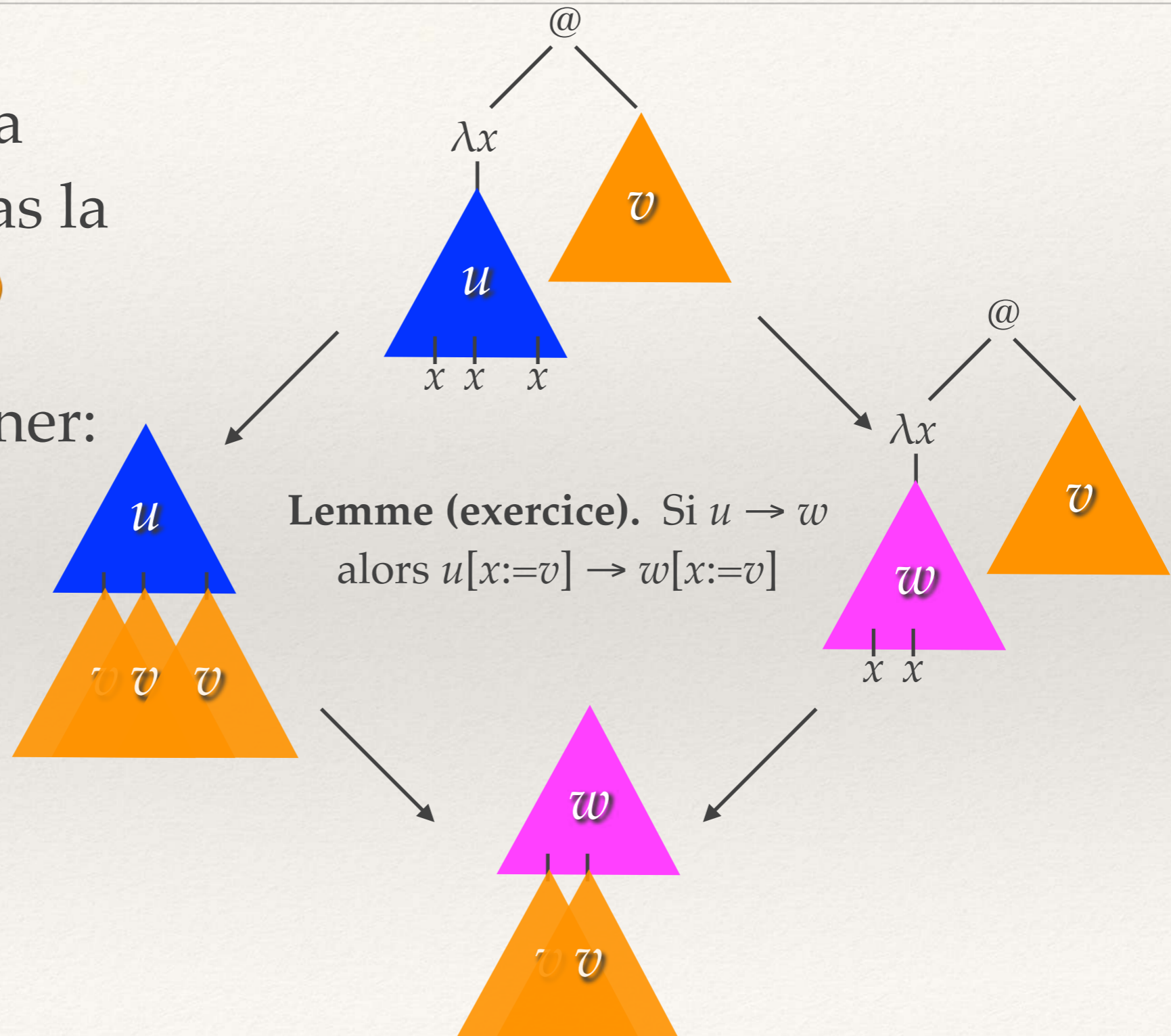
❖ **Cas 2:**

$u \rightarrow w$,

u sous la

λ -abstraction

d'un rédex



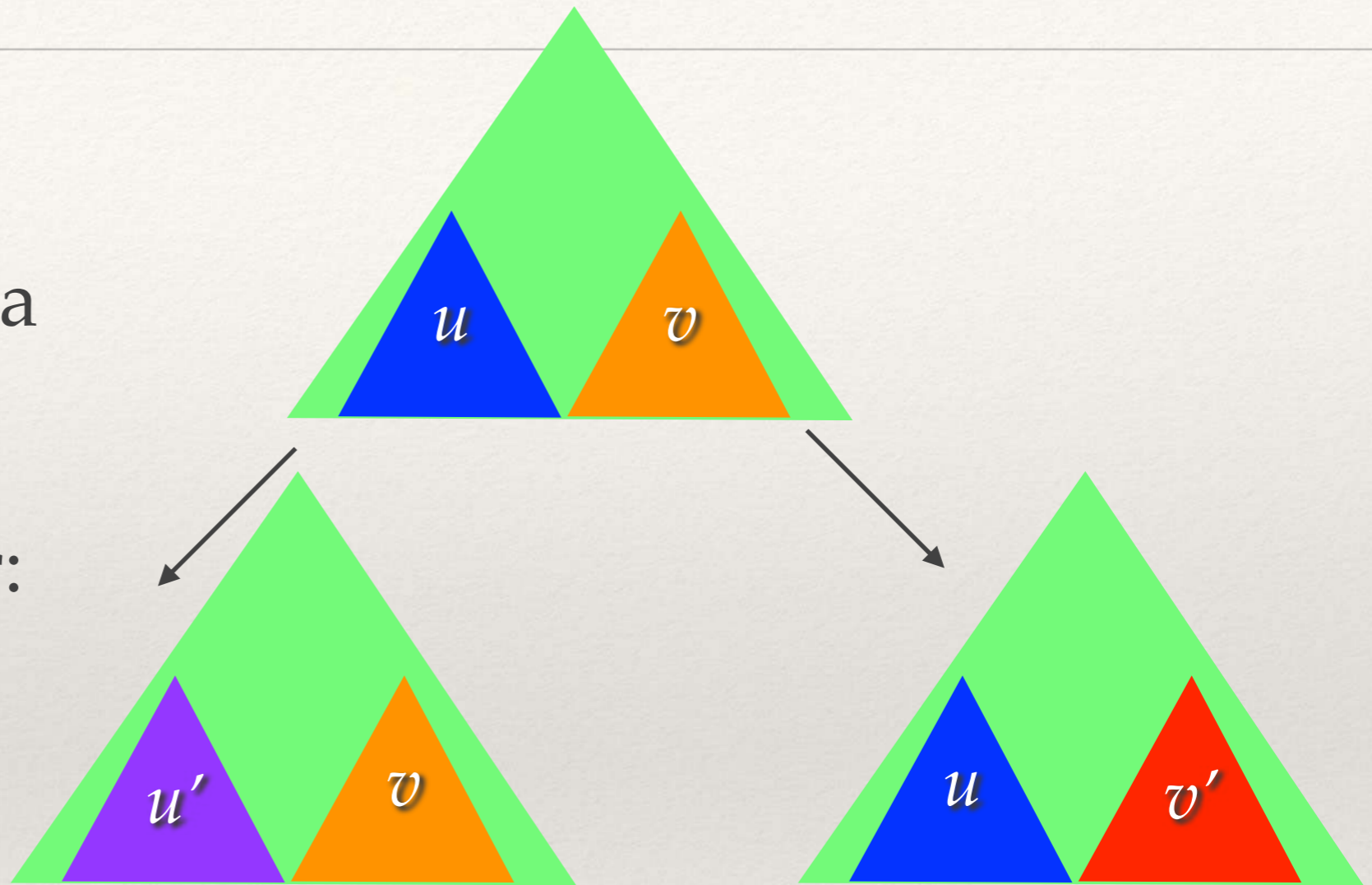
Le λ -calcul est-il localement confluent?

❖ **Oui...** mais ça n'implique pas la confluence 😞

❖ 3 cas à examiner:

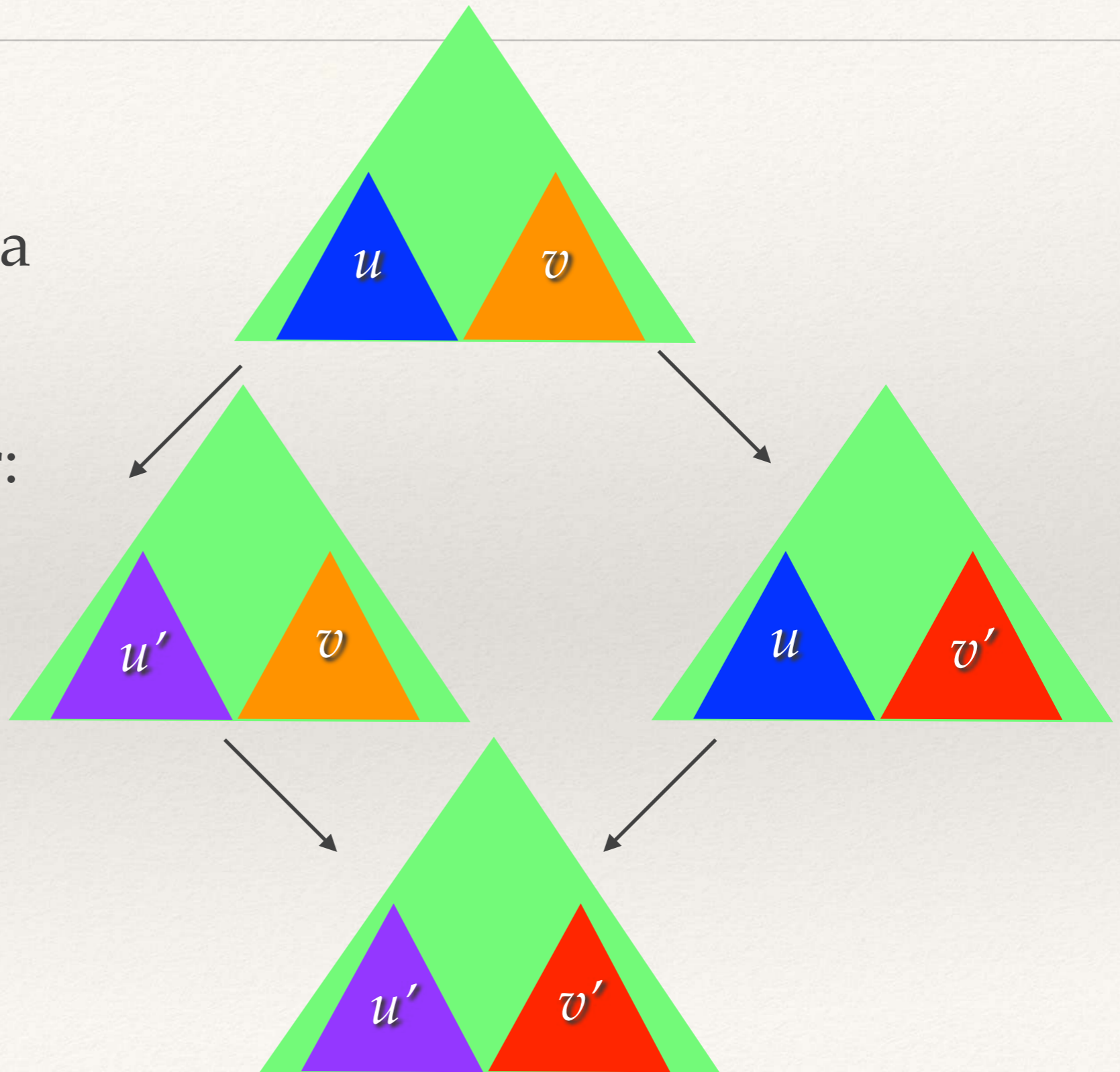
❖ **Cas 3:**
rédex **disjoints**

$u \rightarrow u', v \rightarrow v'$



Le λ -calcul est-il localement confluent?

- ❖ **Oui...** mais ça n'implique pas la confluence 😞
- ❖ 3 cas à examiner:
- ❖ **Cas 3:**
rédex **disjoints**
 $u \rightarrow u', v \rightarrow v'$



Annexe B

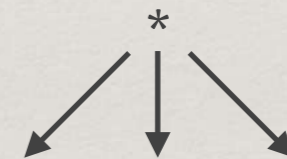
Le lemme de König

- ❖ **Lemme (König).** Tout arbre à branchement fini et dont toutes les branches sont finies ... est fini (n'a qu'un nombre fini de sommets).
- ❖ *Preuve.* Soit T un arbre **infini**, à branchement fini.

*

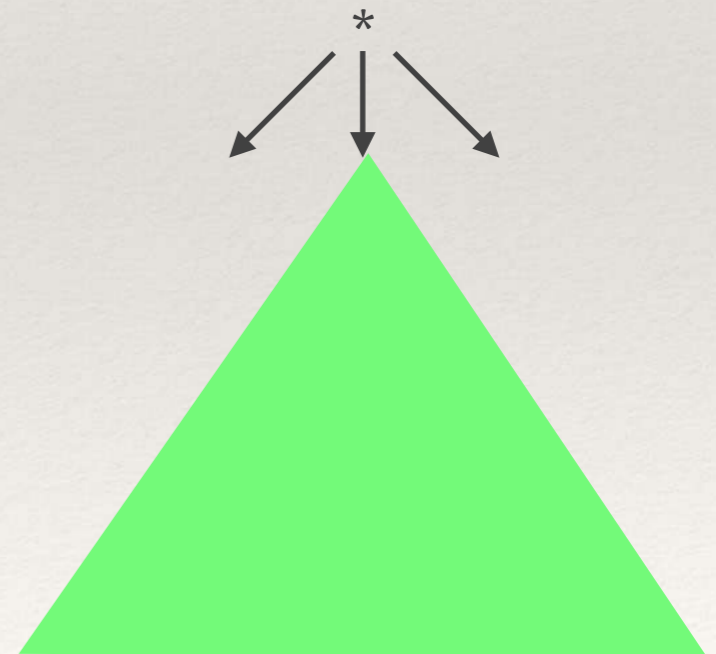
Le lemme de König

- ❖ **Lemme (König).** Tout arbre à branchement fini et dont toutes les branches sont finies ... est fini (n'a qu'un nombre fini de sommets).
- ❖ *Preuve.* Soit T un arbre **infini**, à branchement fini.
- ❖ * n'a qu'un # fini de successeurs



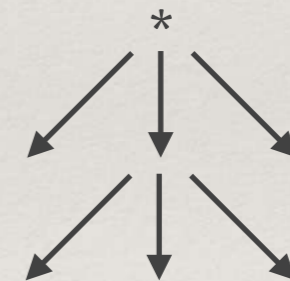
Le lemme de König

- ❖ **Lemme (König).** Tout arbre à branchement fini et dont toutes les branches sont finies ... est fini (n'a qu'un nombre fini de sommets).
- ❖ *Preuve.* Soit T un arbre **infini**, à branchement fini.
- ❖ * n'a qu'un # fini de successeurs
- ❖ **Tiroirs et chaussettes:** un de ceux-là est racine d'un sous-arbre **infini**



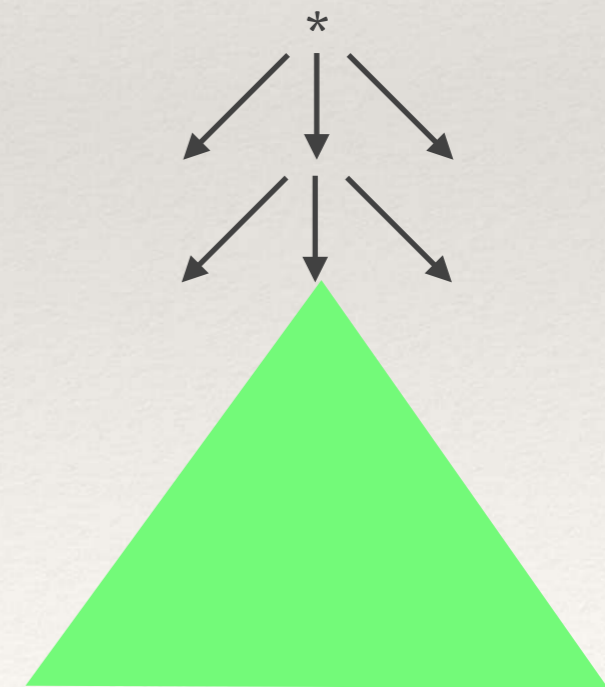
Le lemme de König

- ❖ **Lemme (König).** Tout arbre à branchement fini et dont toutes les branches sont finies ... est fini (n'a qu'un nombre fini de sommets).
- ❖ *Preuve.* Soit T un arbre **infini**, à branchement fini.
- ❖ * n'a qu'un # fini de successeurs
- ❖ **Tiroirs et chaussettes:** un de ceux-là est racine d'un sous-arbre **infini**
- ❖ ... et l'on continue à l'infini, produisant une **branche infinie**.



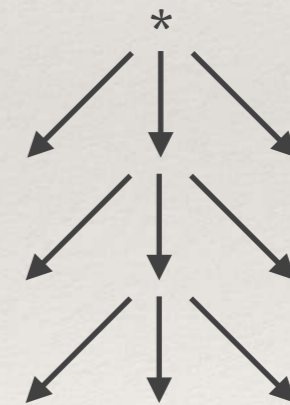
Le lemme de König

- ❖ **Lemme (König).** Tout arbre à branchement fini et dont toutes les branches sont finies ... est fini (n'a qu'un nombre fini de sommets).
- ❖ *Preuve.* Soit T un arbre **infini**, à branchement fini.
- ❖ * n'a qu'un # fini de successeurs
- ❖ **Tiroirs et chaussettes:** un de ceux-là est racine d'un sous-arbre **infini**
- ❖ ... et l'on continue à l'infini, produisant une **branche infinie**.



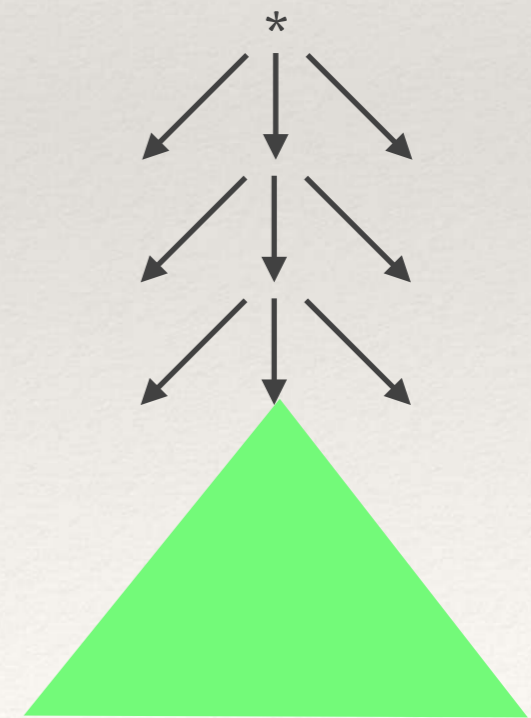
Le lemme de König

- ❖ **Lemme (König).** Tout arbre à branchement fini et dont toutes les branches sont finies ... est fini (n'a qu'un nombre fini de sommets).
- ❖ *Preuve.* Soit T un arbre **infini**, à branchement fini.
- ❖ * n'a qu'un # fini de successeurs
- ❖ **Tiroirs et chaussettes:** un de ceux-là est racine d'un sous-arbre **infini**
- ❖ ... et l'on continue à l'infini, produisant une **branche infinie**.



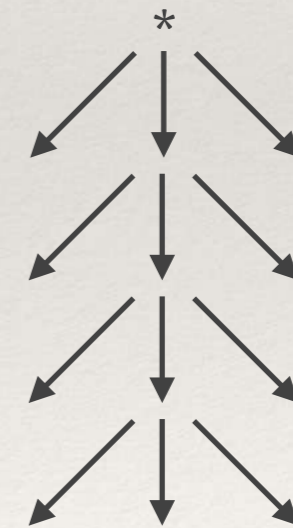
Le lemme de König

- ❖ **Lemme (König).** Tout arbre à branchement fini et dont toutes les branches sont finies ... est fini (n'a qu'un nombre fini de sommets).
- ❖ *Preuve.* Soit T un arbre **infini**, à branchement fini.
- ❖ * n'a qu'un # fini de successeurs
- ❖ **Tiroirs et chaussettes:** un de ceux-là est racine d'un sous-arbre **infini**
- ❖ ... et l'on continue à l'infini, produisant une **branche infinie**.



Le lemme de König

- ❖ **Lemme (König).** Tout arbre à branchement fini et dont toutes les branches sont finies ... est fini (n'a qu'un nombre fini de sommets).
- ❖ *Preuve.* Soit T un arbre **infini**, à branchement fini.
- ❖ * n'a qu'un # fini de successeurs
- ❖ **Tiroirs et chaussettes:** un de ceux-là est racine d'un sous-arbre **infini**
- ❖ ... et l'on continue à l'infini, produisant une **branche infinie**.



Pourquoi $v(u)$ existe-t-il?

- ❖ Supposons \rightarrow fortement normalisable et à **branchement fini**:
pour tout u , $\{v \mid u \rightarrow v\}$ est fini
- ❖ $\forall u$, $v(u) \stackrel{\text{def}}{=} \text{longueur max. d'une réduction partant de } u \text{ existe:}$
on forme l'arbre $T(u) \stackrel{\text{def}}{=} \{v \mid u \rightarrow^* v\}$

Pourquoi $v(u)$ existe-t-il?

- ❖ Supposons \rightarrow fortement normalisable et à **branchement fini**:
pour tout u , $\{v \mid u \rightarrow v\}$ est fini
- ❖ $\forall u$, $v(u) \stackrel{\text{def}}{=} \text{longueur max. d'une réduction partant de } u \text{ existe:}$
on forme l'arbre $T(u) \stackrel{\text{def}}{=} \{v \mid u \rightarrow^* v\}$
- ❖ Il est à branchement fini par hypothèse

Pourquoi $v(u)$ existe-t-il?

- ❖ Supposons \rightarrow fortement normalisable et à **branchement fini**:
pour tout u , $\{v \mid u \rightarrow v\}$ est fini
- ❖ $\forall u$, $v(u) \stackrel{\text{def}}{=} \text{longueur max. d'une réduction partant de } u \text{ existe:}$
on forme l'arbre $T(u) \stackrel{\text{def}}{=} \{v \mid u \rightarrow^* v\}$
- ❖ Il est à branchement fini par hypothèse
- ❖ Ses branches sont finies car \rightarrow fortement normalisable

Pourquoi $v(u)$ existe-t-il?

- ❖ Supposons \rightarrow fortement normalisable et à **branchement fini**:
pour tout u , $\{v \mid u \rightarrow v\}$ est fini
- ❖ $\forall u$, $v(u) \stackrel{\text{def}}{=} \text{longueur max. d'une réduction partant de } u \text{ existe}$:
on forme l'arbre $T(u) \stackrel{\text{def}}{=} \{v \mid u \rightarrow^* v\}$
 - ❖ Il est à branchement fini par hypothèse
 - ❖ Ses branches sont finies car \rightarrow fortement normalisable
- ❖ Par König, $T(u)$ est **fini**, et il n'y a en particulier
qu'un nombre **fini** de réductions partant de u

Pourquoi $v(u)$ existe-t-il?

- ❖ Supposons \rightarrow fortement normalisable et à **branchement fini**:
pour tout u , $\{v \mid u \rightarrow v\}$ est fini
- ❖ $\forall u$, $v(u) \stackrel{\text{def}}{=} \text{longueur max. d'une réduction partant de } u \text{ existe:}$
on forme l'arbre $T(u) \stackrel{\text{def}}{=} \{v \mid u \rightarrow^* v\}$
- ❖ Il est à branchement fini par
- ❖ Ses branches sont finies car
- ❖ Par Kőnig, $T(u)$ est **fini**, et il n'y a qu'un nombre **fini** de ré

Les gens rigoureux dans l'assistance auront remarqué que $T(u)$ est un graphe orienté, pas un arbre... deux solutions:

- (1) montrer que Kőnig reste vrai pour tout **graphe** acyclique à branchement fini
- (2) définir les sommets de $T(u)$ comme les réductions finies partant de u elles-mêmes, poser $^* \stackrel{\text{def}}{=} (u)$ et $p(u \rightarrow u_1 \rightarrow \dots \rightarrow u_{n-1} \rightarrow u_n) = (u \rightarrow u_1 \rightarrow \dots \rightarrow u_{n-1})$

Pourquoi $v(u)$ existe-t-il?

- ❖ Supposons \rightarrow fortement normalisable et à **branchement fini**:
pour tout u , $\{v \mid u \rightarrow v\}$ est fini
- ❖ $\forall u$, $v(u) \stackrel{\text{def}}{=} \text{longueur max. d'une réduction partant de } u \text{ existe}$:
on forme l'arbre $T(u) \stackrel{\text{def}}{=} \{v \mid u \rightarrow^* v\}$

Note (facile mais importante).

Si $u \rightarrow v$ alors $v(u) > v(v)$.

- ❖ Par Kőnig, $T(u)$ est **fini**, et il n'y a qu'un nombre **fini** de réductions

Les gens rigoureux dans l'assistance auront remarqué que $T(u)$ est un graphe orienté, pas un arbre... deux solutions:

(1) montrer que Kőnig reste vrai pour tout **graphe** acyclique à branchement fini

(2) définir les sommets de $T(u)$ comme les réductions finies partant de u elles-mêmes, poser $^* \stackrel{\text{def}}{=} (u)$ et $p(u \rightarrow u_1 \rightarrow \dots \rightarrow u_{n-1} \rightarrow u_n) = (u \rightarrow u_1 \rightarrow \dots \rightarrow u_{n-1})$

*Annexe B: la réduction parallèle est
fortement confluente*

Le lemme de substitution pour \Rightarrow

❖ **Lemme.** Si $u \Rightarrow u'$ et $w \Rightarrow w'$
alors $u[z:=w] \Rightarrow u'[z:=w']$

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

Le lemme de substitution pour \Rightarrow

❖ **Lemme.** Si $u \Rightarrow u'$ et $w \Rightarrow w'$
alors $u[z:=w] \Rightarrow u'[z:=w']$

❖ *Preuve.* Récurrence sur la **taille**

de la dérivation donnée de $u \Rightarrow u'$.

(Exercice. Vous aurez à prouver quelques lemmes auxiliaires.

Attention à α -renommer dans le cas de la règle (β) .)

$$\frac{}{u \Rightarrow u} \quad (0)$$
$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$
$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$
$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

Le lemme de substitution pour \Rightarrow

❖ **Lemme.** Si $u \Rightarrow u'$ et $w \Rightarrow w'$
alors $u[z:=w] \Rightarrow u'[z:=w']$

❖ *Preuve.* Récurrence sur la **taille**

de la dérivation donnée de $u \Rightarrow u'$.

(Exercice. Vous aurez à prouver quelques lemmes auxiliaires.
Attention à α -renommer dans le cas de la règle (β) .)

❖ **Note:** avec la β -réduction (ordinaire), on a:

— si $w \rightarrow w'$ alors $u[z:=w] \rightarrow^* u'[z:=w']$

— si $u \rightarrow u'$ alors $u[z:=w] \rightarrow u'[z:=w]$

$$\frac{}{u \Rightarrow u} \quad (0)$$
$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$
$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$
$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

\Rightarrow est fortement confluente

- ❖ Supposons $s \Rightarrow t_1$ et $s \Rightarrow t_2$
- ❖ On montre qu'il existe t_3 / $t_1, t_2 \Rightarrow t_3$
par récurrence sur **la somme des tailles des dérivations** de $s \Rightarrow t_1$ et $s \Rightarrow t_2$

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

\Rightarrow est fortement confluente

- ❖ Supposons $s \Rightarrow t_1$ et $s \Rightarrow t_2$
- ❖ On montre qu'il existe t_3 / $t_1, t_2 \Rightarrow t_3$
par récurrence sur **la somme des tailles des dérivations** de $s \Rightarrow t_1$ et $s \Rightarrow t_2$
- ❖ A symétrie près, 10 cas

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

	(0)	(β)	(λ)	(@)
(0)				
(β)	—			
(λ)	—	—		
(@)	—	—	—	

\Rightarrow est fortement confluente

- ❖ Supposons $s \Rightarrow t_1$ et $s \Rightarrow t_2$
- ❖ On montre qu'il existe t_3 / $t_1, t_2 \Rightarrow t_3$
par récurrence sur **la somme des tailles des dérivations** de $s \Rightarrow t_1$ et $s \Rightarrow t_2$
- ❖ A symétrie près, 10 cas
- ❖ (0) / - : $s=t_1$ [=u], poser $t_3 \stackrel{\text{def}}{=} t_2$

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

	(0)	(β)	(λ)	(@)
(0)	✓	✓	✓	✓
(β)	-			
(λ)	-	-		
(@)	-	-	-	

\Rightarrow est fortement confluente

- ❖ Supposons $s \Rightarrow t_1$ et $s \Rightarrow t_2$
- ❖ On montre qu'il existe t_3 / $t_1, t_2 \Rightarrow t_3$
par récurrence sur **la somme des tailles des dérivations** de $s \Rightarrow t_1$ et $s \Rightarrow t_2$
- ❖ A symétrie près, 10 cas
- ❖ (0) / - : $s=t_1$ [=u], poser $t_3 \stackrel{\text{def}}{=} t_2$
- ❖ (β) / (λ), (λ) / (@) : impossibles

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

	(0)	(β)	(λ)	(@)
(0)	✓	✓	✓	✓
(β)	-			
(λ)	-	-		
(@)	-	-	-	

\Rightarrow est fortement confluente

- ❖ Supposons $s \Rightarrow t_1$ et $s \Rightarrow t_2$
- ❖ On montre qu'il existe t_3 / $t_1, t_2 \Rightarrow t_3$
par récurrence sur **la somme des tailles des dérivations** de $s \Rightarrow t_1$ et $s \Rightarrow t_2$
- ❖ A symétrie près, 10 cas
- ❖ (0) / - : $s=t_1$ [=u], poser $t_3 \stackrel{\text{def}}{=} t_2$
- ❖ (β) / (λ), (λ) / (@) : impossibles
- ❖ (λ) / (λ), (@) / (@) : par hyp. réc.

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

	(0)	(β)	(λ)	(@)
(0)	✓	✓	✓	✓
(β)	-			
(λ)	-	-		
(@)	-	-	-	

\Rightarrow est fortement confluente

- ❖ Supposons $s \Rightarrow t_1$ et $s \Rightarrow t_2$
- ❖ On montre qu'il existe t_3 / $t_1, t_2 \Rightarrow t_3$
par récurrence sur **la somme des tailles des dérivations** de $s \Rightarrow t_1$ et $s \Rightarrow t_2$

- ❖ A symétrie près, 10 cas
- ❖ (0) / - : $s=t_1$ [=u], poser $t_3 \stackrel{\text{def}}{=} t_2$
- ❖ (β) / (λ), (λ) / (@) : impossibles
- ❖ (λ) / (λ), (@) / (@) : par hyp. réc.
- ❖ (β) / (β) : voir transparent suivant

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

	(0)	(β)	(λ)	(@)
(0)	✓	✓	✓	✓
(β)	-	-	-	-
(λ)	-	-	✓	-
(@)	-	-	-	✓

\Rightarrow est fortement confluente

❖ $(\beta) / (\beta)$: on a

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_1 \quad v \Rightarrow v_1 \end{array}}{\underbrace{(\lambda x . u)}_s \quad v \Rightarrow \underbrace{u_1[x:=v_1]}_{t_1}}$$

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_2 \quad v \Rightarrow v_2 \end{array}}{\underbrace{(\lambda x . u)}_s \quad v \Rightarrow \underbrace{u_2[x:=v_2]}_{t_2}}$$

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

	(0)	(β)	(λ)	(@)
(0)	✓	✓	✓	✓
(β)	—	—	—	—
(λ)	—	—	✓	—
(@)	—	—	—	✓

\Rightarrow est fortement confluente

❖ $(\beta) / (\beta)$: on a

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_1 \quad v \Rightarrow v_1 \end{array}}{\underbrace{(\lambda x . u)}_s \quad v \Rightarrow \underbrace{u_1[x:=v_1]}_{t_1}}$$

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_2 \quad v \Rightarrow v_2 \end{array}}{\underbrace{(\lambda x . u)}_s \quad v \Rightarrow \underbrace{u_2[x:=v_2]}_{t_2}}$$

❖ Par h.r., on a $u_1, u_2 \Rightarrow u_3$
et $v_1, v_2 \Rightarrow v_3$

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

	(0)	(β)	(λ)	(@)
(0)	✓	✓	✓	✓
(β)	—	—	—	—
(λ)	—	—	✓	—
(@)	—	—	—	✓

\Rightarrow est fortement confluente

❖ $(\beta) / (\beta)$: on a

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_1 \quad v \Rightarrow v_1 \end{array}}{\underbrace{(\lambda x . u)}_s \quad v \Rightarrow \underbrace{u_1[x:=v_1]}_{t_1}}$$

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_2 \quad v \Rightarrow v_2 \end{array}}{\underbrace{(\lambda x . u)}_s \quad v \Rightarrow \underbrace{u_2[x:=v_2]}_{t_2}}$$

❖ Par h.r., on a $u_1, u_2 \Rightarrow u_3$
et $v_1, v_2 \Rightarrow v_3$

❖ Par le...
 $t_1 \Rightarrow u_3[x:=v_3]$

Lemme. Si $u \Rightarrow u'$ et $v \Rightarrow v'$
alors $u[x:=v] \Rightarrow u'[x:=v']$

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

	(0)	(β)	(λ)	(@)
(0)	✓	✓	✓	✓
(β)	—	—	—	—
(λ)	—	—	✓	—
(@)	—	—	—	✓

\Rightarrow est fortement confluente

❖ $(\beta) / (\beta)$: on a

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_1 \quad v \Rightarrow v_1 \end{array}}{\underbrace{(\lambda x . u)}_s \quad v \Rightarrow \underbrace{u_1[x:=v_1]}_{t_1}}$$

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_2 \quad v \Rightarrow v_2 \end{array}}{\underbrace{(\lambda x . u)}_s \quad v \Rightarrow \underbrace{u_2[x:=v_2]}_{t_2}}$$

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

❖ Par h.r., on a $u_1, u_2 \Rightarrow u_3$
et $v_1, v_2 \Rightarrow v_3$

❖ Par le... **Lemme.** Si $u \Rightarrow u'$ et $v \Rightarrow v'$
alors $u[x:=v] \Rightarrow u'[x:=v']$
 $t_1 \Rightarrow u_3[x:=v_3]$

❖ Par le... **Lemme.** Si $u \Rightarrow u'$ et $v \Rightarrow v'$
alors $u[x:=v] \Rightarrow u'[x:=v']$
 $t_2 \Rightarrow u_3[x:=v_3]$

	(0)	(β)	(λ)	(@)
(0)	✓	✓	✓	✓
(β)	—	—	—	—
(λ)	—	—	✓	—
(@)	—	—	—	✓

\Rightarrow est fortement confluente

❖ $(\beta) / (\beta)$: on a

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_1 \quad v \Rightarrow v_1 \end{array}}{\underbrace{(\lambda x . u)}_s \quad v \Rightarrow \underbrace{u_1[x:=v_1]}_{t_1}}$$

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_2 \quad v \Rightarrow v_2 \end{array}}{\underbrace{(\lambda x . u)}_s \quad v \Rightarrow \underbrace{u_2[x:=v_2]}_{t_2}}$$

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

❖ Par h.r., on a $u_1, u_2 \Rightarrow u_3$
et $v_1, v_2 \Rightarrow v_3$

❖ Par le... **Lemme.** Si $u \Rightarrow u'$ et $v \Rightarrow v'$
alors $u[x:=v] \Rightarrow u'[x:=v']$
 $t_1 \Rightarrow u_3[x:=v_3]$

❖ Par le... **Lemme.** Si $u \Rightarrow u'$ et $v \Rightarrow v'$
alors $u[x:=v] \Rightarrow u'[x:=v']$
 $t_2 \Rightarrow u_3[x:=v_3]$

	(0)	(β)	(λ)	(@)
(0)	✓	✓	✓	✓
(β)	—	✓	—	—
(λ)	—	—	✓	—
(@)	—	—	—	✓

C'est le t_3 désiré

\Rightarrow est fortement confluente

❖ Dans le (dernier) cas (β) / $(@)$, on a:

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_1 \quad v \Rightarrow v_1 \end{array}}{\underbrace{(\lambda x . u) v}_s \Rightarrow \underbrace{u_1[x:=v_1]}_{t_1}}$$

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ (\lambda x . u) \Rightarrow w \quad v \Rightarrow v_2 \end{array}}{\underbrace{(\lambda x . u) v}_s \Rightarrow \underbrace{wv_2}_{t_2}}$$

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

	(0)	(β)	(λ)	(@)
(0)	✓	✓	✓	✓
(β)	—	✓	—	—
(λ)	—	—	✓	—
(@)	—	—	—	✓

\Rightarrow est fortement confluente

❖ Dans le (dernier) cas (β) / $(@)$, on a:

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_1 \quad v \Rightarrow v_1 \end{array}}{\underbrace{(\lambda x . u) v}_s \Rightarrow \underbrace{u_1[x:=v_1]}_{t_1}} \quad \frac{\begin{array}{c} \vdots \\ \vdots \\ (\lambda x . u) \Rightarrow w \quad v \Rightarrow v_2 \end{array}}{\underbrace{(\lambda x . u) v}_s \Rightarrow \underbrace{wv_2}_{t_2}}$$

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

❖ Mais $(\lambda x . u) \Rightarrow w$ ne peut être dérivé que par (0) ou (λ) ... donc la situation est...

	(0)	(β)	(λ)	$(@)$
(0)	✓	✓	✓	✓
(β)	—	✓	—	—
(λ)	—	—	✓	—
$(@)$	—	—	—	✓

\Rightarrow est fortement confluente

❖ Dans le (dernier) cas (β) / $(@)$, on a:

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_1 \quad v \Rightarrow v_1 \end{array}}{\underbrace{(\lambda x . u)}_s \quad \underbrace{v \Rightarrow u_1[x:=v_1]}_{t_1}}$$

$$\frac{\begin{array}{c} \vdots \\ u \Rightarrow u_2 \\ \vdots \\ (\lambda x . u) \Rightarrow (\lambda x . u_2) \quad v \Rightarrow v_2 \end{array}}{\underbrace{(\lambda x . u)v}_s \Rightarrow \underbrace{(\lambda x . u_2)v_2}_{t_2}}$$

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

	(0)	(β)	(λ)	(@)
(0)	✓	✓	✓	✓
(β)	—	✓	—	—
(λ)	—	—	✓	—
(@)	—	—	—	✓

\Rightarrow est fortement confluente

❖ Dans le (dernier) cas (β) / $(@)$, on a:

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_1 \quad v \Rightarrow v_1 \end{array}}{\underbrace{(\lambda x . u) v}_s \Rightarrow \underbrace{u_1[x:=v_1]}_{t_1}}$$

$$\frac{\begin{array}{c} \vdots \\ u \Rightarrow u_2 \\ \vdots \\ (\lambda x . u) \Rightarrow (\lambda x . u_2) \quad v \Rightarrow v_2 \end{array}}{\underbrace{(\lambda x . u) v}_s \Rightarrow \underbrace{(\lambda x . u_2) v_2}_{t_2}}$$

❖ Par h.r., on a $u_1, u_2 \Rightarrow u_3$
et $v_1, v_2 \Rightarrow v_3$

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

	(0)	(β)	(λ)	(@)
(0)	✓	✓	✓	✓
(β)	—	✓	—	—
(λ)	—	—	✓	—
(@)	—	—	—	✓

\Rightarrow est fortement confluente

❖ Dans le (dernier) cas (β) / $(@)$, on a:

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_1 \quad v \Rightarrow v_1 \end{array}}{\underbrace{(\lambda x . u) v}_s \Rightarrow \underbrace{u_1[x:=v_1]}_{t_1}}$$

$$\frac{\begin{array}{c} \vdots \\ u \Rightarrow u_2 \\ \vdots \\ (\lambda x . u) \Rightarrow (\lambda x . u_2) \quad v \Rightarrow v_2 \end{array}}{\underbrace{(\lambda x . u)v}_s \Rightarrow \underbrace{(\lambda x . u_2)v_2}_{t_2}}$$

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

❖ Par h.r., on a $u_1, u_2 \Rightarrow u_3$
et $v_1, v_2 \Rightarrow v_3$

❖ Par le... **Lemme.** Si $u \Rightarrow u'$ et $v \Rightarrow v'$
alors $u[x:=v] \Rightarrow u'[x:=v']$

	(0)	(β)	(λ)	(@)
(0)	✓	✓	✓	✓
(β)	—	✓	—	—
(λ)	—	—	✓	—
(@)	—	—	—	✓

\Rightarrow est fortement confluente

❖ Dans le (dernier) cas (β) / $(@)$, on a:

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_1 \quad v \Rightarrow v_1 \end{array}}{\underbrace{(\lambda x . u) v}_s \Rightarrow \underbrace{u_1[x:=v_1]}_{t_1}}$$

$$\frac{\begin{array}{c} \vdots \\ u \Rightarrow u_2 \\ \vdots \\ (\lambda x . u) \Rightarrow (\lambda x . u_2) \quad v \Rightarrow v_2 \end{array}}{\underbrace{(\lambda x . u)v}_s \Rightarrow \underbrace{(\lambda x . u_2)v_2}_{t_2}}$$

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

❖ Par h.r., on a $u_1, u_2 \Rightarrow u_3$
et $v_1, v_2 \Rightarrow v_3$

❖ Par le... **Lemme.** Si $u \Rightarrow u'$ et $v \Rightarrow v'$
 $t_1 \Rightarrow u_3[x:=v_3]$ alors $u[x:=v] \Rightarrow u'[x:=v']$

❖ Par (β) ,
 $t_2 \Rightarrow u_3[x:=v_3]$

	(0)	(β)	(λ)	(@)
(0)	✓	✓	✓	✓
(β)	—	✓	—	—
(λ)	—	—	✓	—
(@)	—	—	—	✓

\Rightarrow est fortement confluente

❖ Dans le (dernier) cas (β) / $(@)$, on a:

$$\frac{\begin{array}{c} \vdots \\ \vdots \\ u \Rightarrow u_1 \quad v \Rightarrow v_1 \end{array}}{\underbrace{(\lambda x . u) v}_s \Rightarrow \underbrace{u_1[x:=v_1]}_{t_1}}$$

$$\frac{\begin{array}{c} \vdots \\ u \Rightarrow u_2 \\ \vdots \\ (\lambda x . u) \Rightarrow (\lambda x . u_2) \quad v \Rightarrow v_2 \end{array}}{\underbrace{(\lambda x . u)v}_s \Rightarrow \underbrace{(\lambda x . u_2)v_2}_{t_2}}$$

$$\frac{}{u \Rightarrow u} \quad (0)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{(\lambda x . u) v \Rightarrow u'[x:=v']} \quad (\beta)$$

$$\frac{u \Rightarrow u'}{\lambda x . u \Rightarrow \lambda x . u'} \quad (\lambda)$$

$$\frac{u \Rightarrow u' \quad v \Rightarrow v'}{uv \Rightarrow u'v'} \quad (@)$$

❖ Par h.r., on a $u_1, u_2 \Rightarrow u_3$
et $v_1, v_2 \Rightarrow v_3$

❖ Par le... **Lemme.** Si $u \Rightarrow u'$ et $v \Rightarrow v'$
alors $u[x:=v] \Rightarrow u'[x:=v']$

❖ Par (β) ,
 $t_2 \Rightarrow u_3[x:=v_3]$

C'est le t_3 désiré

	(0)	(β)	(λ)	(@)
(0)	✓	✓	✓	✓
(β)	—	✓	—	✓
(λ)	—	—	✓	—
(@)	—	—	—	✓